1-1-2001

VLSI implementation of a massively parallel wavelet based zerotree coder for the intelligent pixel array

Geoffrey N. Alagoda
Edith Cowan University

2001

VLSI implementation of a massively parallel wavelet based zerotree coder for the intelligent pixel array

Geoffrey N. Alagoda
Edith Cowan University

Edith Cowan University

Copyright Warning

VLSI Implementation of a Massively Parallel Wavelet Based Zerotree Coder for the Intelligent Pixel Array

By
GEOFFREY NISHANTHA ALAGODA

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Doctor of Philosophy
at the
School of Engineering & Mathematics
Edith Cowan University

Principal Supervisor: Professor Kamran Eshraghian
Associate Supervisor: Dr. Alexander Rassau

Nov 2001

To my fiancé Vanisha,

my father Gamini,

my mother Deanna,

and my brother Dale.

Contents

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Declaration

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education; and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signature . Date 5/11/2002

Acknowledgements

I would like to express my gratitude to the following people at Edith Cowan University.

Prof. Kamran Eshraghian, my principal supervisor for his invaluable guidance, support and encouragement throughout this study. Kamran always inspires enthusiasm in those that he works with. This thesis would have never been completed without the financial support that he procured.

Dr. Alexander Rassau, my secondary supervisor, for his time, patience, encouragement and effort correcting this thesis. Alex's supervision was vital in the timely completion of the thesis through his insightful knowledge of the topic.

Dr. Stefan Lachowicz, who provided the initial supervision, for his suggestions, time and encouragement.

Andrew Ehrhardt, David Lucas, Joe Austin-Crowe, Vishalakshi Ramakonar and **Edward Gluszak,** for their input, comradeship, and encouragement.

I would also like to express my gratitude to the following institutions.

School of Engineering, Edith Cowan University for all the financial and equipment support provided during the course of this study. **Intelligent Pixels Inc.** for financial and software tools support, and the **Australian Research Council (ARC),** for providing the numerous grants that supported this study and the research team.

Family support was without question one of the most vital factors influencing the completion of the research. They motivated and encouraged me unfailingly, so thanks go to my father **Gamini,** my mother **Deanna** and my brother **Dale.**

One person who was always there to pull me through the tough times, and always had faith in my ability to achieve this goal, was my fiancé **Vanisha Nair.** My deepest gratitude goes to her.

ABSTRACT

In the span of a few years, mobile multimedia communication has rapidly become a significant area of research and development constantly challenging boundaries on a variety of technological fronts. Mobile video communications in particular encompasses a number of technical hurdles that generally steer technological advancements towards devices that are low in complexity, low in power usage yet perform the given task efficiently. Devices of this nature have been made available through the use of massively parallel processing arrays such as the Intelligent Pixel Processing Array. The Intelligent Pixel Processing array is a novel concept that integrates a parallel image capture mechanism, a parallel processing component and a parallel display component into a single chip solution geared toward mobile communications environments, be it a PDA based system or the video communicator wristwatch portrayed in "Dick Tracy" episodes. This thesis details work performed to provide an efficient, low power, low complexity solution surrounding the massively parallel implementation of a zerotree entropy codec for the Intelligent Pixel Array.

Bandwidth limitations that surround many of today's communications channels have forced investigations into viable and efficient compression algorithms for application in mobile video communications. However the complexity presented by most modern hybrid image / video compression algorithms typically reserve them for pure software based implementations. Recent trends in the field indicate that a hybrid video compression algorithm involving a discrete wavelet transform coupled together with a zerotree entropy coder provides an effective compromise between compression efficiency and complexity. However, most solutions still remain purely software based.

Zerotree coding is an efficient lossless entropy coding technique designed for image and video compression. Its strength lies in its ability to generate single symbol representations for large areas of transformed coefficients. However, the computational complexity the zerotree coder presents, generally reserves it to a purely software based solution typically requiring hardware that is less efficient in power, such as DSPs. The low-power usage of the Intelligent Pixel Processing Array therefore becomes an attractive alternative. The solution to the problem of efficiently performing zerotree coding in a massively parallel

pixel based architecture is thus provided herein. A novel pixel-wise zerotree self-classification technique is used to convert the sequential zerotree codec to its parallel equivalent. Two zerotree coding algorithms were investigated, the Embedded Zerotree Wavelet (EZW) and the Zerotree Entropy Coder (ZTE), with the latter being chosen for the final design, due to its suitability for implementation within a parallel processing environment.

The Intelligent Pixel Array is composed of an N x M matrix of pixel based processing elements. The aim of the work performed here is for the development of a QCIF sized processing array containing (176 x 144) identical pixel elements. The successful implementation of this concept within a maximum possible die size of around 15mm x 15 mm forces the pixel pitch to revolve around the size of 80µm x 80 µm. This area is then subdivided into areas for the capture and ADC, forward and inverse motion compensation processing, forward and inverse wavelet transform processing, forward and inverse zerotree coding and image display. Therefore the zerotree solution provided by this thesis overcomes two essential problems. 1. A pixel based parallel processing division of the predominantly sequential zerotree coding algorithm and 2. The integration of the zerotree codec into the shared pixel pitch of 80 x 80 µm, using 0.25µm technology.

The prototype designed as a result of this work implements a 32 x 32 pixel array, consisting of approximately 1 million transistors and consuming less than 35mW of power at a low operating frequency of 100 KHz, while still coding at 25 frames per second. The fully scalable nature of this architecture allows for any arbitrary sized implementation of this array, with minimal or no modification to each pixel element.

Following the first introductory chapter, this thesis proceeds onto a description of current image / video compression methodologies. A full background to, and comparison of, the two zerotree coding techniques are provided next. The subsequent chapter introduces the concept of the massively parallel Intelligent Pixel Array and prior work performed on such a device. A parallel hardware architecture is then presented for each zerotree coding technique, and the more suitable technique is chosen. Next physical layouts and hardware simulations are presented for the chosen solution implemented in 0.25µm CMOS technology. Finally a conclusions and directions towards future work in this area are provided.

Publications

The following is a list of papers published during the course of this thesis development.

A. M. Rassau, G. Alagoda, D. Lucas, J. Austin-Crowe, K. Eshraghian (1999) – "Massively Parallel Intelligent-Pixel Implementation of a Zerotree Entropy Video Codec for Multimedia Communications", *VLSI: Systems On A Chip*, Kluwer Academic Publishers, Portugal, 89-100.

A. M. Rassau, G. Alagoda, K. Eshraghian (1999) – "Massively Parallel Wavelet Based Video Codec For An Intelligent-Pixel Mobile Multimedia Communicator", *Fifth International Symposium on Signal Processing and its Application*, Queensland, 793-795.

H. N. Cheung, G. Alagoda, K. Eshraghian and L. Ang (1999) – "Smart Pixel VLSI Architecture For Embedded Zerotree Wavelet Coding", *Fifth International Symposium on Signal Processing and its Application*, Queensland, 693-695.

A. M. Rassau, R. Mavaddat, G. Alagoda and K. Eshraghian (1999) – "System Analysis of An Intelligent Pixel Mobile Multimedia Communicator", *Fifth International Symposium on Signal Processing and its Application*, Queensland, 801-803.

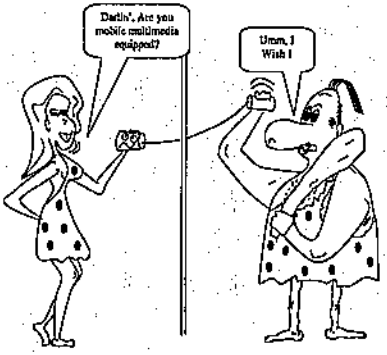
G. Alagoda, A. M. Rassau and K. Eshraghian (2001) – "A Massively Parallel Per-Pixel Based Zerotree Processing Architecture for Real-Time Video Compression", *to be published in SPIE's International Symposium on Microelectronics and Micro-Electro-Mechanical Systems*, Adelaide.

Figures

Tables

Chapter 1

MOBILE MULTIMEDIA COMMUNICATIONS SYSTEMS



1.1. Introduction

Humankind's never-ending thirst for more information and more reliable services has been a powerful driving force behind nearly all advancements in communication technology to date. Communication systems, from its humble beginnings of message scrolls, carrier pigeons, smoke clouds etc., have evolved significantly through the advent of new

technologies throughout time. Advancements in electrical and particularly electronic technologies have revolutionised the means providing communications today.

Traditionally, communications systems were generally comprised of a single form of media, voice, written letters, etc. Since the advent of the television, which supported two forms of media (audio and video), increasing demands for devices capable of transmitting multiple forms of media, has resulted. The increasing complexity and decreasing size of electronics components, particularly in the area of digital signals processing, coupled together with advancements in wireless communication standards, have introduced the concepts of compact mobile communications devices, such as mobile phones. Recent mobile devices and communications standards allow for communication via multiple media, such as voice, text and pictures. This leads us to the present day where mobile multimedia devices, which can support voice, text, faxes, web browsing, high resolution pictures, and full motion video, appear just around the corner. Figure 1-1 illustrates a third generation (3G) mobile multimedia communications concept as proposed by Nokia.

Figure 1-1: 3G Multimedia Communications Concept

Integrating full motion video capability into a mobile multimedia communications device presents a number of significant challenges. These include standards development, network infrastructure development, high compression efficiency and low power operation of circuitry. Solutions to all of these problems are not suggested in this thesis, as it defines

too large a scope. This thesis, however, aims to explore the hardware (VLSI) development of an efficient video compression codec focused for applications in multimedia communication devices.

Transmission of 'raw' or uncompressed video, even a simple black image, unfortunately, consumes a large volume of data bandwidth. For instance, considering a greyscale video sequence consisting of 352 pixels by 288 pixels at 25 frames a second (video quality); it consumes approximately 20 megabits per second or the equivalent of approximately 6758 voice transmissions per second. Even though technological advances are constantly increasing the capacity of modern communication channels, bandwidth is generally not cheap. Images and video, in 'raw' format tend to contain a significant amount of redundancy, be it in the form of large singular coloured areas, low frequency spatial gradients or transmission of similar content from frame to frame. Video compression algorithms today, such as MPEG 1/2/4, H261/263 and MJPEG2000, minimise, not only the spatial redundancy in frames, but also the similarities found between two consecutive frames, therefore, allowing for low bandwidth representations of the original video sequence. The techniques employed to do so generally dictate the compression efficiency of the codec.

A typically video compression system is composed of the components illustrated in Figure 1-2. Example algorithms belong to each component category, particularly those employed in this thesis, is listed below each relevant component. The highlighted (grey) components, however, constitute the major focus of this thesis.

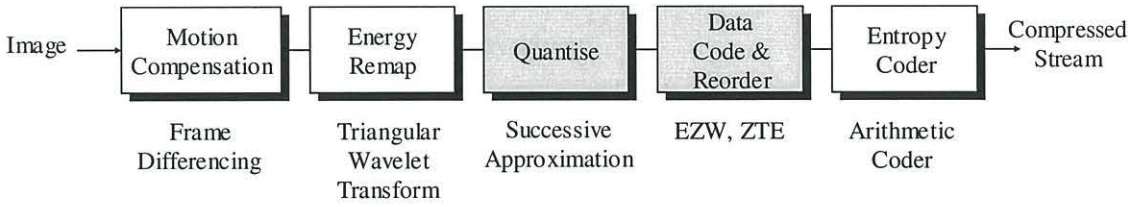


Figure 1-2: Video Compression Components

In December 1993, Jerome M. Shapiro, introduced an image compression scheme based on the wavelet transform and a novel coefficient mapping technique, which he named the Embedded Zerotree Wavelet (EZW). In September 1996, Stephen Martucci and Iraj Sodagar also proposed a modified version of the EZW, which was targeted more towards

low bit-rate video coding and was named the ZeroTree Entropy coder (ZTE). Both proposed algorithms provided an efficient technique to map coefficients in multiple wavelet subbands into trees which could be represented using a single symbol. Thus exploiting the self-similarities between subbands, which are inherent to coefficients resulting from a wavelet transform. The major differences between the two include,

1. **Different symbol sets** – The EZW uses a 4 symbol set, while the ZTE uses a simpler 3 symbol set.
2. **Quantisation method** – The EZW contains an embedded quantisation algorithm while the ZTE uses an external subband quantisation process.
3. **Number of passes** – The number of passes vary significantly as the EZW uses multiple passes while the ZTE only two.

Although, the EZW coder was initially proposed for image compression its usefulness is not limited to this, and like the ZTE coder, it too can be employed for compression of video sequences. In this thesis, designs for both video codec are presented, particularly in regards to the hardware realisation of a device suitable for mobile multimedia applications.

1.2. Motivation of Thesis

The motivations behind this thesis are derived from two perspectives,

1.2.1. Mobile Multimedia Communications Industry

"Watching films and concerts anywhere

By 2005, we will be downloading films, live concerts, and games through the air, enabling us to access the entertainment of our choice anytime and anywhere. We will carry a new type of slim-line digital communications device to enable us to relax and work more efficiently on the move. Pay-per-access services will mushroom, and the film and music industries will increasingly use mobile multimedia services to test-market products."

A future prediction by a company called Roke, owned by Siemens.
http://www.roke.co.uk/news/archive/science_fiction.htm

Comments such as this, although made with a light-heart, predict some of the future possibilities for mobile multimedia communications, and humankind's impending need to embrace these concepts in daily life. Compact mobile communications equipment were once only a gleam in the eyes of the science fiction writer (such as Gene Roddenberry – for the creation of captain Kirk's communicator), yet today, there are an estimated 553 million mobile customers globally. Figure 1-3 illustrates the growth of the mobile phone user group since 1992, as taken from,

http://home.intekom.com/cellular/.

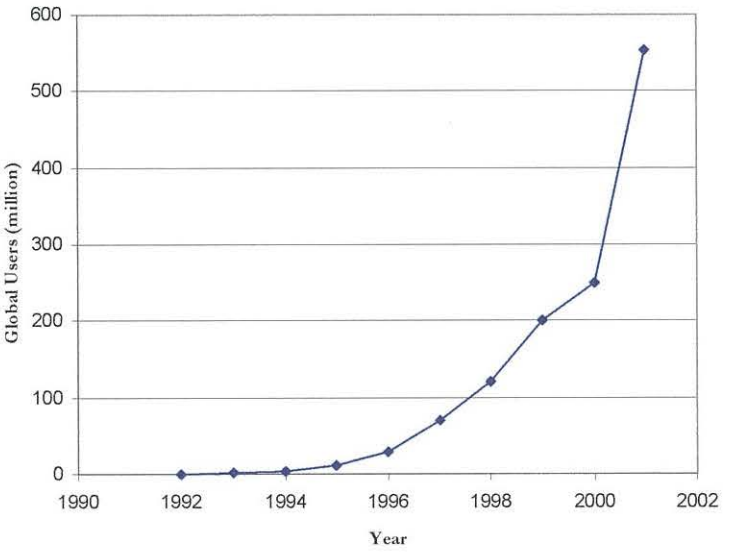


Figure 1-3: World-wide mobile phone users

In addition to this, since its introduction, SMS messaging, another form of media, has taken off to result in the exchange of nearly 15 billion global messages monthly (December 2000) and it is still increasing. More information is available at http://uk.gsmbox.com/news/mobile_news/all/30904.gsmbox

These figures indicate that, a significant number of global communities have not only fully embraced mobile multimedia communication technology but are also willing to expand usage to other forms of communication media if the opportunity exists. Communication via video, therefore, has the potential to fully blossom in tomorrow's market and as such significantly motivates research into this area.

1.2.2. Novel Research Opportunity

Mobile multimedia communications applications tend to demand products that consume little development resources (e.g. high bandwidth interfacing), are of compact size and consume minimal power. To target these demands the designs this thesis have adopted a more unified capture, display and processing approach than that used in conventional modular systems. This proposed unified approach is advantageous because, as far as the end product designer is concerned, it virtually eliminates the high-bandwidth interfacing required between the capture, processing and display components, and as such potentially reduces the associated development cost, in comparison to a modular system. In addition to this, a massively parallel processing approach has also been adopted. The concurrency inherent to parallel processing arrays enable them to provide the same functionality as a single processor system, yet at a reduced clock frequency. This reduction in clock speed, has the potential to reduce the power requirements of the device, rendering it more suitable for mobile multimedia products.

Both of the design choices made above present some difficult, yet interesting challenges particularly when considering implementing in VLSI devices. Several groups were setup to produce solutions to these problems and were organised into the structure illustrated in Figure 1-4.

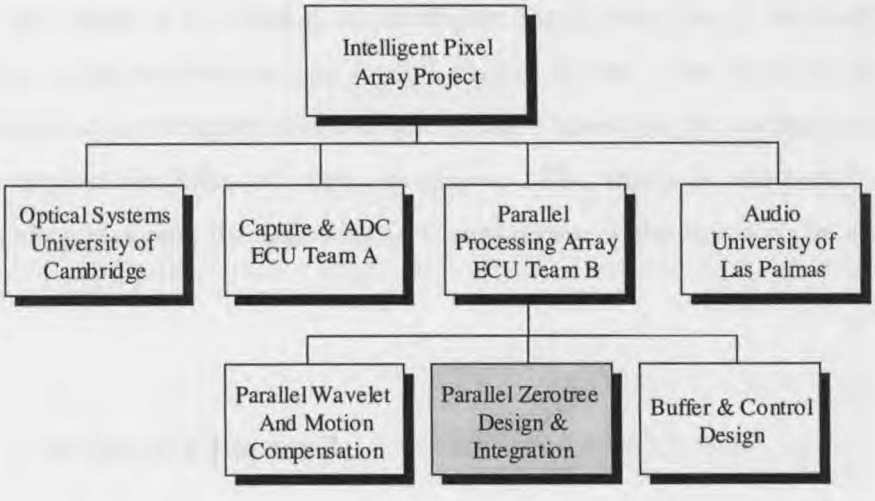


Figure 1-4: Task Allocation Structure

The research motive for this thesis relates to the design and realisation of a fully integrated massively parallel zerotree codec (the highlighted blocks) for the Intelligent Pixel Array Processor. Some of the challenges this thesis aims to undertake include,

1. Conversion of the serial processor based zerotree algorithms to massively parallel pixel based algorithm.
2. Facilitate a highly parallel zerotree significant search mechanism in VLSI hardware.
3. Design architectures for both EZW and ZTE algorithms.
4. Design self-classification mechanisms.
5. Design array load and unload mechanisms.
6. Design scale and status based pixel activation mechanisms.
7. Integration of both codecs into a parallel wavelet transform architecture.
8. Compare the performance and complexity issues of both EZW and ZTE codecs.
9. Produce a fully custom layout of the selected codec in 0.25 micron VLSI technology.

1.3. Scope of Thesis

The aim of this thesis is to develop an Intelligent Pixel Array based massively parallel zerotree video coding architecture and its full custom layout. Two zerotree (EZW and ZTE) algorithms are investigated and a choice is made based on the coding performance and implementation feasibility of both algorithms. The thesis is composed of seven chapters including this one; the following is a breakdown of the scope to be covered by each.

1.3.1. Scope of Chapter 2

Chapter 2 is an introductory chapter which presents several concepts related to images, video and video compression. The concepts introduced include explanations of images, video, image sampling, image/video compression, test images used and codec

components such as, motion analysis and compensation (Block Matching Motion Estimation, frame differencing), image transforms (wavelets, Discrete Cosine Transform), image quantisation (subband) image/video coding schemes (zigzag, run-length) and entropy coding (Variable Length Coding, arithmetic). This chapter presents the basics for usage in other chapters.

1.3.2. Scope of Chapter 3

Chapter 3 presents an in-depth investigation into the two, Embedded Zerotree Wavelet (EZW) and ZeroTree Entropy (ZTE), image/video coding zerotree algorithms. This chapter covers the concepts of zerotree symbol generation, significance identification of wavelet coefficients, subband quantisation, successive approximation, zerotree codec components, tree search algorithms and the image coding performance of both algorithms. The chapter introduces the terminology used in chapters to follow.

1.3.3. Scope of Chapter 4

Chapter 4 firstly introduces the Intelligent Pixel Array and Intelligent Pixel paradigms and their application to communications systems. Secondly it presents the modified architectures designed to support both the wavelet transform component and the zerotree codec. The chapter specifically covers the concepts of pixel-wise capture display and processing infrastructure, scale control, high-pass/low-pass pixel identification structure, pixel communication, the primary register component, motion compensation components and pixel mode architecture. The component architectures suggested here are used in conjunction with the zerotree architectures to constitute the full video codec.

1.3.4. Scope of Chapter 5

Chapter 5 presents the full parallel hardware architectures for both the EZW and ZTE algorithms. A comparison, in terms of compression efficiency and hardware complexity, is then presented to select a particular codec for prototype implementation. The chapter covers the areas of NBLOCK format, the parallel search interconnections,

significance identification, symbol generation, data load and unload architectures and edge buffer specifications. Both codec efficiencies are tested on two image sequences and a hardware comparison is also made. Finally, the better hardware design is selected.

1.3.5. Scope of Chapter 6

Chapter 6 presents the full custom layout for the ZTE architecture in a 0.25µm UMC CMOS process, for the IP100P prototype. A simulated power calculation is then made. The chapter covers the areas of specifics relating to the design technology, the basic gates used, the buffer drivers, the zerotree components, power distribution, pin distribution and the entire array layout. HSpice simulations are made on the completed pixel design to obtain the necessary power measures.

1.3.6. Scope of Chapter 7

Chapter 7 presents a conclusion to the thesis and also presents some future research options for continuation of the work.

Chapter 2

IMAGE/VIDEO CODING TECHNIQUES

"A picture is worth a thousand words."[1]

Fred Barnard *(circa 1920).*

2.1. Introduction

A viewpoint on this slogan suggests that a description of a detailed scene quite often necessitates the use of significantly large number of words. Words, which are limited in individually descriptive content, when assembled together as a part of the whole, can describe a natural scene in all its detail. Similarly, a digital representation of a detailed image requires a large number of its primary detail components, namely bits. A typical greyscale image (PAL CIF) of 352 x 288 pixels, with 8 bits of colour per pixel, requires 792 kilobits to be represented in full. Images with larger resolutions and/or more colours (HDTV, DVD) require significantly more space to be fully represented. Given an environment with a large to infinite bandwidth, the transmission of these images for communications purposes presents few obstacles. However, most real-world communications applications are limited in available bandwidth and as such a compromise between the detail levels and the data rate (bps) is applicable to virtually all visual communications systems. For instance most modern modems with data rates in the range 15 – 56 kbps or ISDN channels that use 64 kbps or the implementations of the new 384

[1] According to a paper [1] published by Alan F. Blackwell a picture is actually worth 84.1 words.

kbps 3G [2] standards are still well under the required bandwidth for 25 frame per second QCIF (176 x 144) video communications streams, which consumes around 5Mbps. To circumvent the lack of bandwidth in most communication channels, video communications systems in general eliminate redundant information from the stream while attempting to represent the unique detail content with an acceptable compromise between image-quality and bit-rate. One that performs as such, can be considered as an efficient video compression system for a particular type of video sequence, since the efficiency of a video coder is directly related to factors such as video content, required video quality, frames per second, coder / decoder complexity etc.

Video is composed of one of two basic types of images; natural image (Figure 2-1) or animated sequences be it computer generated or hand drawn (Figure 2-2).

Figure 2-1: Natural Image (Jenny)

Figure 2-2: Generated Image

Clearly Figure 2-1, typical of most photographs or video camera sequences, consists of more colours and colour gradients than Figure 2-2, and as such it tends to compress better with an energy remapping coder such as Joint Picture Experts Group (JPEG) [3] [4], Wavelets or Set Partitioning In Hierarchical Trees (SPIHT). In comparison Figure 2-2, typical of generated art or bi-level images from faxes, clearly contain more solid colour areas and sharper edges, and as such it is better compressed with coders such as JBIG or GIF. See [5] for more details on the comparisons. Generated art tends to exhibit greater higher frequency properties and also contain large single valued image areas; which typically and efficiently coded by entropy coding techniques such as run-length coding, variable length coding, arithmetic coding etc. Natural image compression algorithms in comparison tend to be complex in nature than their counterparts, because they employ transforms and quantisation schemes in addition to the entropy codecs [6]. For all intents and purposes the transform and quantisation stage attempts to represent the image in a "generated art" manner to be efficiently coded by an entropy coder. The complexity of a natural image codec is generally a reflection of its efficiency as shown in [7] where the more complex SPIHT codec out performs the generic JPEG codec. This thesis will focus on a natural image compression scheme, as the targeted application, namely mobile multimedia communication, depends heavily on a codec that performs efficient natural video compression for low-bit rate communications.

Video can be considered as a set of similar images that vary in content at a specific rate in time (eg @ 25 fps for PAL). Therefore a typical video compression technique provides a means of efficiently coding temporal redundancy between successive images in addition to a general image-coding algorithm. Some major hybrid video/image compression standards available today, as presented in [5] are listed in Table 2-1.

Hybrid video coding techniques today, tend to employ at least four key compression stages, as indicated in Figure 2-3, with each stage uniquely contributing to the compression process. The following section will cover some aspects of each of the stages.

Table 2-1: Image Video Compression Standards

Standard	Creating Body	Target Application	Type	Year
CCITT G3	*ITU-T*	*Facsimile*	*Image*	*1980*
CCITT G4	*ITU-T*	*Facsimile*	*Image*	*1984*
GIF	*Compuserve*	*Generic Indexed Pictures*	*Image*	*1987/89*
H.261	ITU-T	Video over ISDN (64Kbps)	Video	1990
JPEG	*Joint Picture Experts Group (ISO, ITU-T, IEC)*	*Generic Natural Pictures*	*Image*	*1992*
MPEG-1	Motion Picture Experts Group (ISO, IEC)	Video On CD	Video	1992
JBIG	*Joint Bi-level Image Group (ISO, ITU-T, IEC)*	*Binary Images*	*Image*	*1993*
J.81	ETSI	Video Distribution Over Public Networks	Video	1994
MPEG-2 / H.262	Motion Picture Experts Group (ISO, ITU-T, IEC)	Generic Interlaced Video	Video	1995
H.263 / G.723.1	ITU-T	Video Over POTS	Video	1996
MPEG-4	Motion Picture Experts Group (ISO, ITU-T, IEC)	Object Based Video / Images	Image / Video	1998
MPEG-7	Motion Picture Experts Group (ISO, ITU-T, IEC)	Multimedia Content Discription Interface	Misc	2001

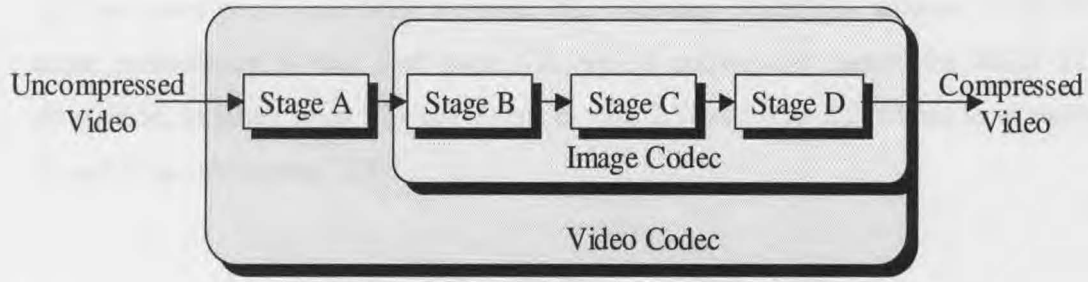


Figure 2-3: Image / Video Coding Stages

Stage A – Temporal Redundancy Compensation

Stage B – Image Energy Remap or Transform

Stage C – Quantisation or Lossy Compression

Stage D – Entropy Coding or Non-lossy Compression

2.2. Image / Video Coding Primitives

2.2.1. RGB – YUV Colour Space

Images captured, typically for compression and display, are represented in one of two standards, either RGB (Red Green Blue) or YUV (Luminance and Chrominance) [8]. The RGB technique is often used for reproduction or capture, while the YUV is more often used for processing and transmission. Images captured in RGB typically use 24 bits per pixel with an 8-bit value for each colour component in the pixel, resulting in around 16 million possible colour levels per pixel. The main disadvantage of the RGB scheme is the lack of a common intensity component, which restricts the processing on the image to processing performed on each individual colour. In comparison the YUV

scheme employs one luminance component (Y) and two chrominance components (UV) to represent the image. The luminance component can be realised as a greyscale representation of the image, which when supplemented by the two chrominance components, represents the original colour image. The fundamental advantage of this representation originates from human perception of a colour image, where defects in the detail levels in the intensity component affect the visual stimulus significantly more in comparison to defects in colour [9]. Therefore, by maintaining the intensity component to a higher degree of accuracy a compromise can be made on the detail levels of the luminance components. The three components are represented using 8-bits per pixel per component; however the sampling scheme is adjusted to remove some redundancy at this first stage. A typical conversion matrix for RGB YUV translation, as taken from [8] can be seen in (Eq. 2.1) and (Eq. 2.2) (After level shifting U and V by subtracting 128).

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1678 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

(Eq. 2.1)

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

(Eq. 2.2)

Codecs that use the YUV standard readily allow for greyscale images as well as colour, and as such this standard is adopted for the rest of this document.

2.2.2. YUV Image Sampling

Since the human eye is less susceptible to errors in colour as compared to intensity, the first basic compression mechanism is usually applied here. This is achieved by sub-sampling the colour components further. The usual sampling ratio between Y:U:V as employed in standards such as MPEG, H.261 etc. is fixed at 4:1:1 (a 4:2:2 scheme is also available where only the horizontal chrominance component is reduced). In this

scheme each of the chrominance components are sampled at half the sampling rate of the luminance component in each direction, resulting in a four-fold decrease in the two colour component pixels, in comparison to the RGB scheme. The 4:1:1 sampling pattern pictured in Figure 2-4 corresponds to an equivalent 12 bits per pixel, that "appears" very close to the 24-bit per pixel RGB scheme. However each sample in each colour component is still converted to an 8-bit value.

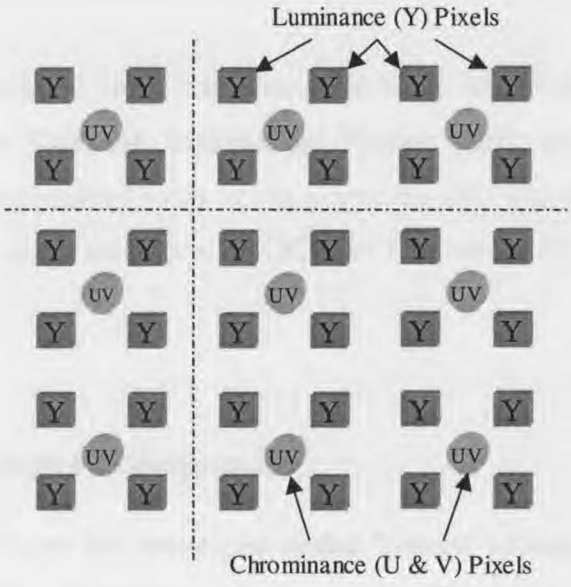


Figure 2-4: Luminance & Chrominance sample positioning

2.2.3. Image / Video Resolutions

The sizes of images that are captured vary in resolution dramatically from one application to another, usually ranging anywhere from 32x32 to any arbitrary resolution, with the occasional exception to either dimension being a multiple of 8. Influenced by television systems, video codec resolutions however, have adopted a number of default resolution standards. As referenced in [5] [8] the most common formats are listed in Table 2-2. The size of the image influences the computational complexity of a particular algorithm and in practice larger images compress to a better ratio than smaller images.

Table 2-2: Common Video Resolutions

Format	Y	UV	Usage
QCIF - NTSC	176 x 120	88 x 60	Teleconferencing
QCIF - PAL	176 x 144	88 x 72	Teleconferencing
CIF - NTSC	352 x 240	176 x 120	Teleconferencing / Video
CIF - PAL	352 x 288	176 x 144	Teleconferencing / Video
SIF - 525	720 x 480	360 x 480	Movies, High Res.Video
SIF - 625	720 x 576	360 x 576	Movies, High Res. Video

The two main standards used for teleconferencing and video communications applications are, the Common Intermediate Format (CIF) and the Quarter CIF (QCIF). Since the applications focus of this document slots into the category of video communication the image resolution of QCIF or CIF with PAL are used for codec testing purposes.

2.2.4. Test Images & Sequences

As a test image, the 'Lena' (or sometimes spelled 'Leanna') image, is one of the most commonly used images in the available literature. The image exhibits both sharp cornered textures and gradual changers in texture, while providing a human face as a test image. The image is square and generally found in resolutions ranging from 1024 x 1024 down to around 32 x 32. The image used here is 256 x 256 x 8 bits for the greyscale and 256 x 256 x 24 bits for the colour, as this resolution somewhat resembles the QCIF standard. The greyscale image and its colour version can be seen in Figure 2-5.

As far as video sequences are concerned, two different sequences are used in this document. One the proprietary 'Jenny' sequence and the other the standard 'Salesman' Sequence.

Figure 2-5: Lena Y & RGB Images

The 'Jenny' sequence was filmed on a CCD camera, within the post graduate engineering labs at Edith Cowan University. It portrays a woman, Jenny, speaking as if over a video phone. The camera was hand held to provide a sequence that emulated a mobile video phone device and as such is susceptible to hand held tremors. The clip was shot at 25 fps PAL with CIF (352 x 288) resolution and is composed of 24-bit colour. The sequence is 62 seconds long and is composed of 1557 frames. The sequence was later down sampled to QCIF (176 x 144) and converted to 6-bit greyscale to suit the application in this document. The colour and greyscale versions of the first frame in the Jenny sequence are presented in Figure 2-6.

Figure 2-6: The 'Jenny' Sequence

The 'Salesman' sequence is a professionally shot fixed camera sequence of a salesman busy plying his trade. The sequence contains a large number of high detail and contrasting objects, e.g. books, which remain stationary throughout the sequence. The only moving component is the salesman. This is a standard sequence used by many

video compression algorithm developers to test their codecs. The original sequence was captured at 25 fps and at a resolution of 360 x 288 pixels. The sequence is in 8-bit greyscale and is 449 frames (18 seconds) long. For test purposes in this application, the sequence was then sub sampled to QCIF (176 x 144) resolution, using 6-bit greyscale. Figure 2-7 illustrates the first frame of the salesman sequence.

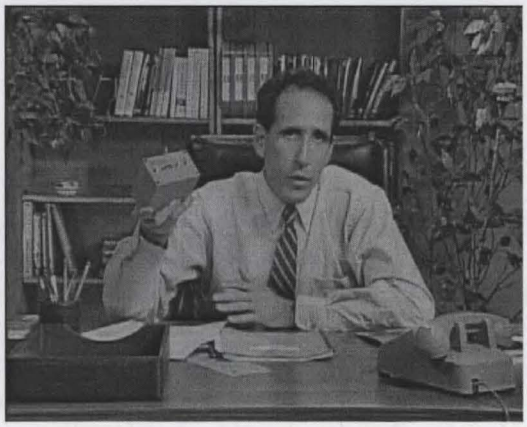


Figure 2-7: The 'Salesman' Sequence

2.3. Motion Analysis (Stage A)

Motion estimation, prediction and compensation, collectively known as motion analysis, are performed on video sequences to minimise the temporal redundancy associated with successive images which contain a small spatial variance between frames [10]. A common example video clip, which tends to have very minimal temporal variance, typically portrays an individual reading the news in a stationary background. The only movement in such cases tend to be limited to the individual's face, eyes, lips and hands. Continuous retransmission of frames from such a sequence generally results in the excessive use of bandwidth, as the successive frames will only contain minimum changes from the first. Therefore, motion analysis is employed to code the changing areas only. There are three concepts surrounding the area of motion analysis, as extrapolated from [8] [11], which have the following meanings.

2.3.1. Motion Estimation

Motion Estimation is generally considered as the process of detecting movement between two consecutive frames in a sequence, and calculating a motion vector to denote its displacement. MPEG [12] and H.263 [13] standards employ this technique

to perform block motion estimation. The motion vector, υ, is typically calculated for a 16 by 16 pixel block having both X and Y coordinates. If $n = (N_x, N_y)$ represents a pixel at the head of a 16 by 16 block at time τ with intensity I_τ and at time τ -1 the head pixel was at $o = (O_x, O_y)$, then the vector υ is generated in such a manner to minimise the error between the block intensities I_τ and $I_{\tau-1}$. This way the new block intensity at position n could be generated from the contents at o and shifted to position $o + \upsilon$. Figure 2-8 illustrates the use of these vectors [15].

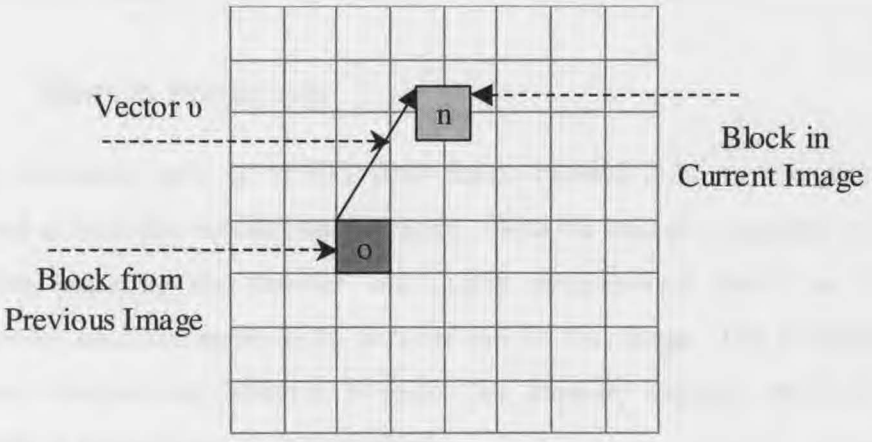


Figure 2-8: Motion Estimation

A function calculating the sum of absolute difference (SAD) (Eq. 2.3) between two blocks of data is classically used to calculate the error between a block in the current frame and a block in the previous frame. For each block in the current frame this function is repeated on several blocks in the previous frame while searching for the minimal displacement. The search distance is usually set to 16 pixels in either direction from the origin, however standards such as MPEG-2 [14] have provisions for longer and non-integral search distances to support variable sized blocks [16].

$$SAD = \sum (N - O)^2$$

(Eq. 2.3)

Where N and O represent the current block and previous block respectively.

2.3.2. Motion Compensation

Since the motion estimation process only searches for the minimum SAD value below retransmit threshold, the possibility that non-zero SAD values existing is not zero. This implies that the motion estimation process located the closest match and not an identical match. To compensate for this a difference block, $N - O_r$ is transmitted. This process is termed *motion compensation* [17]. Frame differencing is an extreme case where motion compensation is performed on the entire frame from one frame to the next and hence a simpler hardware alternative to full motion analysis.

2.3.3. Motion Prediction

Used in standards such as MPEG (IBP frame system) [12], motion prediction is performed at both the encoder and decoder. Here the encoder attempts to track the predictions made by the decoder and supply compensated blocks to cover the difference between that made by the decoder and the real image. This technique allows for better compression, however increases the memory capacity required and the complexity of the coding / decoding scheme.

When performing motion analysis three different frame types are employed; Intra, Inter and Predicted frames. The Intra-frames are fully encoded frames that provide a base for coding Inter-frames and Predicted-frames, and as such require a larger bandwidth than the other two. The Inter-frames and Predicted-frames contain the motion estimation components. Intra-frames are regularly injected to 'refresh' any deviation caused by Inter or Predicted frames within a compressed sequence. In this manner optimal results are attained.

2.4. Discrete Cosine Transform (Stage B)

The Discrete Cosine Transform (DCT), first developed in [18], is the most commonly used 2D orthogonal image transform present today. Standards such as JPEG, MPEG 1/2/4, H.261/3 etc. employ this transform to relocate the spatial energy in the image into a few coefficients. This transform behaves in a manner approximating a real-valued Fourier Transform, packing the significant coefficients into the lower frequency areas of the transformed spatial positions [19]. This also has the effect of de-correlating the

dependencies between adjacent pixels in the time domain. Therefore, an image containing numerous sharp texture edges produces a DCT with a high number of larger magnitude coefficients throughout the spectrum. If the number of high magnitude coefficients were to be reduced or quantised then the inverse transform redistributes this loss over the entire image eliminating the high frequency sharp edges generating a blurred image.

The lowest frequency coefficient of a DCT is termed the DC coefficient as it represents the average image level. This value generally is an order of magnitude larger than the next largest coefficient and is normally left un-quantised, as it degrades the reconstructed image significantly. Figure 2-10 illustrates a typical DCT performed on a 16 x 16 image of Lena's eye (Figure 2-9), showing the compaction of energy into the low frequency spectrum. Note in Figure 2-10 the DC coefficient was reduced by a factor of 10 to allow for display.

Figure 2-9: Lena's Eye

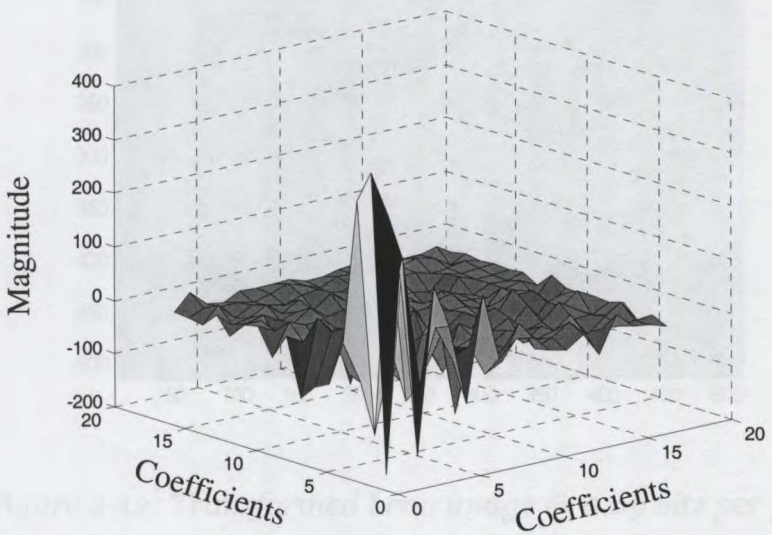


Figure 2-10: 16 x 16 & 2D DCT Transform of eye

To illustrate the effect of quantisation on a transformed image, Figure 2-11 illustrates an image that has been quantised at the extreme level in the time domain to generate bi-level image at 1 bit per pixel. While in Figure 2-12 the image has been quantised at an even lower 0.69 bits per pixel after undergoing a DCT and still remains near perfect.

Figure 2-11: "Raw" Lena image @ 1 bit per pixel

Figure 2-12: Transformed Lena image @ 0.69 bits per pixel

The standard forward and inverse 2D DCT can be mathematically expressed as in (Eq. 2.4) and (Eq. 2.5) respectively [21].

$$F(u,v) = (1/4)C(u)C(v)\sum_{i=0}^{M}\sum_{j=0}^{N} f(i,j)\cos\left(\frac{(2i+1)u\pi}{2M}\right)\cos\left(\frac{(2j+1)v\pi}{2N}\right), \begin{matrix} 0 \le u \le (M-1) \\ 0 \le v \le (N-1) \end{matrix}$$

$$C(x) = \begin{cases} 1/\sqrt{2} & x = 0 \\ 1 & \text{otherwise} \end{cases}$$

(Eq. 2.4)

$$f(i,j) = (1/4)\sum_{u=1}^{M-1}\sum_{v=1}^{N-1} C(u)C(v)F(u,v)\cos\left(\frac{(2u+1)i\pi}{2M}\right)\cos\left(\frac{(2v+1)j\pi}{2N}\right), \begin{matrix} 1 \le i \le M \\ 1 \le j \le N \end{matrix}$$

(Eq. 2.5)

Where

$F(u,v)$ = the transformed coefficient.

$f(i,j)$ = The image values.

M x N = The size of block.

However in most cases, by exploiting the orthogonal nature of the DCT, a 2D DCT can be realised by performing multiple single dimensional transforms on each of the rows and columns. Furthermore to ease the computational burden the DCT convolution poses an integer DCT evaluation has been presented in [20]. The forward and inverse single dimensional transforms are realised by evaluating (Eq. 2.6) and (Eq. 2.7) for N number of samples [18] [21].

$$F(u) = C(u)\sum_{i=0}^{N-1} f(i)\cos\left(\frac{(2i+1)\pi u}{2N}\right), \quad 0 \le u \le (N-1)$$

(Eq. 2.6)

$$f(i) = C(u)\sum_{u=0}^{N-1} F(u)\cos\left(\frac{(2u+1)\pi i}{2N}\right), \quad 0 \le i \le (N-1)$$

(Eq. 2.7)

Where,

$$C(x) = \begin{cases} \sqrt{\frac{1}{N}} & , x = 0 \\ \sqrt{\frac{2}{N}} & , 1 \le x \le (N-1) \end{cases}$$

Standards such as H.261, H.263 MPEG 1, 2 & 4 which actively use the DCT as part of the video processing algorithm, segment the image into a number of smaller 8 x 8 time domain blocks before these block are transformed [13]. The advantage here is the reduction in complexity and register resources as needed to perform a full image transform, especially in hardware. In addition the transform can usually be performed in parallel on all the blocks in the image. The disadvantage to this technique is that under higher levels of compression needed for lower bandwidth usage, it suffers from blocking artefacts

Figure 2-13: 'Lena' DCT (JPEG) @ 0.2 bits per pixel

Figure 2-13 illustrates the 'Lena' image after it has undergone an 8 x 8 block transform and quantisation to represent 0.2 bits per pixel (bpp). The blocking artefacts are caused as a

result of the spectral truncation that occurs when quantisation is applied to the frequency components of a block. This affects the spatial image by driving all the pixels in a block to the same colour level; hence two adjacent blocks that have different "average" frequencies will contrast against each other resulting in a blocking effect.

A full image transform will avert the blocking artefacts, however since the DCT is not ideally suited for full image transforms, due to the hardware complexity that such an implementation requires, simpler techniques such as wavelets and zerotrees are generally used to perform full image transforms. [7] [22]

2.4.1. Matrix Based 2D DCT Computation

The 2D DCT can be computed, for parallel implementations, employing a matrix based approach. If *R*, is a vector from *0* to *(R-1)* and *Co* is a vector from *0* to *(C-1)*, where *R* and *C* represent the row and column width of the block to be transformed, also given *CM* to be a $R \times C$ matrix containing the cosine values generated by (Eq. 2.8), then computing the statements in Figure 2-14 results in generation of a 2D DCT for that block. Where F is f transformed. The algorithm for the inverse is as in Figure 2-15.

$$CM = \cos\left(\left(\overline{Ro} \times \left(Co + \frac{1}{2}\right)\right)\pi C\right) \qquad \textit{(Eq. 2.8)}$$

1. $CM1 = CM(R,C)$
2. $CM2 = CM(C,R)$
3. $F = CM2 \times \left(\overline{CM1 \times f}\right)$

Figure 2-14: Forward 2D DCT Algorithm

1. $CM1 = CM(R,C)$
2. $CM2 = CM(C,R)$
3. $CM1(1,:) = 0.5 \times (CM1(1,:))$
2. $CM2(1,:) = 0.5 \times (CM2(1,:))$
3. $f = CM2 \times \left(\overline{CM1 \times F}\right)$

Figure 2-15: Inverse 2D DCT Algorithm

2.5. Discrete Wavelet Transform (Stage B)

Since its first theoretical formulation by J. Morlet, A. Grossmann [26] and Y. Meyer [27], wavelet theory has been applied with effective results to a variety of signal processing applications in the fields of data compression, image compression, image processing, time-frequency spectral estimation etc.[25]. Daubechies [28] and Mallat [29] conducted the pioneering work bridging the gap between the wholly mathematical wavelet and its digital signals processing counterpart. A predominant feature of wavelets and wavelet analysis, by which it facilitates digital signals processing, relies in its ability to effectively represent a signal in its time-frequency equivalent [30]. This time-frequency conversion property of the DWT, remaps a signal's transient components to a position on the time-frequency plane, which represents the predominant frequency of that component at the particular time of its occurrence. Hence each coefficient in the transform is determined by taking an inner product between the input function and a suitably chosen wavelet basis function. This value then represents, in some sense, the degree of similarity between the input function and that particular basis function [31]. If the basis functions are orthogonal (or orthonormal), then an inner product taken between two basis functions is zero, indicating that these are all completely dissimilar. Therefore, if the input image is composed of components that are similar to one, or a few, of the basis functions, then all but one, or a few, of the coefficients will result in relatively small values.

The application of wavelets to images for compression purposes stems from the analysis of typical images, which tend to indicate that even though areas of large spatial activity are easily identified, extensive areas of low spatial variance or substantially uniform areas can be identified as well. Since the rapidity of change in spatial variance can be expressed in terms of spatial frequency components, [32] suggested that typical images have strong low-frequency components that necessitate preservation. Given that a DWT represents spatial components in time-frequency space, when applied to an image this results in the separation of the vertical and horizontal spatial frequency components into multi-resolution subbands. Figure 2-16 depicts the 'Lena' image in multiple subbands each at a different resolution. An excellent comparison between different wavelet based subband transforms are presented in [33].

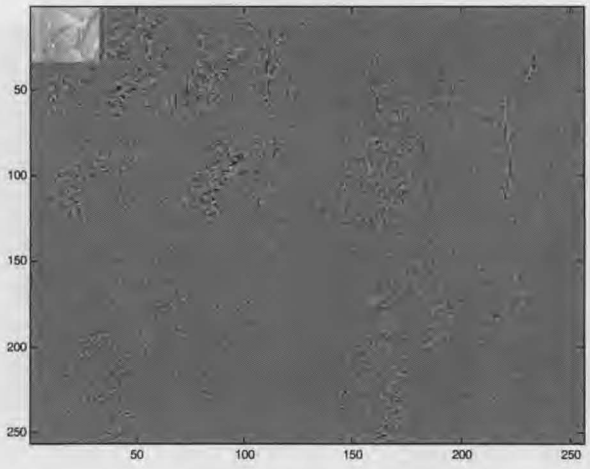


Figure 2-16: 'Lena' image in multi-resolution subbands

When this concept is applied to a 2D image and a suitable quantisation and entropy coding scheme is applied on the generated subbands, the result has shown [22] to outperform most DCT based compression systems, especially at low bandwidths. At lower compression levels, however, where quantisation methods above 1 bpp are used, [34] has shown that DCT based schemes with the same source coder results in improved image quality. The main tradeoffs between the very common DCT based systems and the Wavelet based systems lie between the complexity of implementation and efficiency. Wavelet based schemes often provide a more efficient compression scheme at low cost in complexity.

To obtain an efficient compression rate, once the image is transformed into the form of Figure 2-16, a psycho-visual quantisation scheme is applied in varying degrees to the different subbands, with the aim of minimising the possible significant coefficient count. Figure 2-17 shows the 'Lena' image quantised at 0.19 bpp using a DWT method and a relative entropy encoder. It exhibits an obvious improvement in visual quality over the 0.2 bpp DCT image in Figure 2-13

Figure 2-17: DWT of 'Lena' image @ 0.19 bpp

2.5.1. Filter Based Wavelet Decomposition

The purely mathematical approach to performing a subband decomposition of a 2D image in theory involves the computation of the inner products of a basis mother wavelet and components of the image [35]. This however, in terms of computational complexity, is quantifiably comparable to performing a standard tried and proven DCT, if not higher. Fortunately a technique introduced by [36] has shown that 2D wavelet decomposition is easily approximated by solely performing a selected set of simple filter functions on the image. This fast DWT technique realises the subband decomposition by iteratively performing a two-band subdivision of the low-frequency image components into high and low pass frequency segments. Since the greatest frequency component in the filtered sequence is bound to half that of the original, both filtered streams can be sub-sampled by two without being subjected to aliasing effects. Figure 2-18 illustrates a typical subband decomposition scheme for the iterative realisation of a one-dimensional DWT by employing a high / low pass filters and coefficient down sampling.

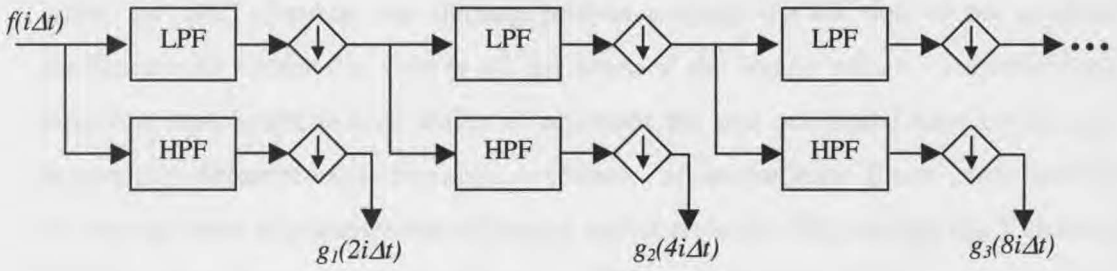


Figure 2-18: DWT subband decomposition via filters

The maximum number of iterations that can be performed is related to the size of the data vector being transformed and is described by (Eq. 2.9). Where N is the number of elements in the array; is not odd and is exactly divisible by 2^I. Where *floor(x) is* a function that returns *x* truncated to *0* decimal places.

$$I = floor\left(\log_2(N)\right)$$

(Eq. 2.9)

Literature tends to suggest that the use of three, four or five iterations or scales provides a good balance between performance and complexity [37]. Reconstruction of the image is just the reapplication of similar filters followed by an up-sampling and interpolation process conducted on both sets of data that are then re-combined via a summation process.

The key to performing an effective DWT and its inverse is heavily dependant on the quality of selected transform filters [39]. Most modern wavelet transform techniques employ Quadrature Mirror Filters (QMF) for the two-band filters, as aliasing caused by the forward transform while using such filters, are cancelled in the inverse transform hence providing near perfect reconstruction [38]. Furthermore, by choosing an orthogonal set of filters, a 2D transform is easily realised by alternatively performing multiple single dimensional transforms on the rows and columns. The subband subdivision that results from a 2D wavelet transform is illustrated in Figure 2-19. The like colours in the figure represent the frequency components, increasing in spatial frequency as the size increases, for the horizontal, vertical and the diagonal. Figure 2-20 illustrates the first two decompositions of an image using a 2D transform. An

interesting attribute of the DWT is that it can be easily performed on an entire image unlike the DCT, because the filtering process restricts the bit size of the resultant coefficients to within the vicinity of the sizes of the image values. Imperfections, however, exist in the lack of ability to represent the real numbered filter coefficients precisely in different implementation scenarios. Some common filters used, listed in decreasing order of compression efficiency and complexity [40], include the Villasenor 9-7 filter, the Daubechies 8 tap filter, the Villasenor 2-6 binary filter, the triangular binary filter and the Haar binary filter. These filters belong to the category of QMFs, where Table 2-3 refers to the standard coefficients used to evaluate these filters.

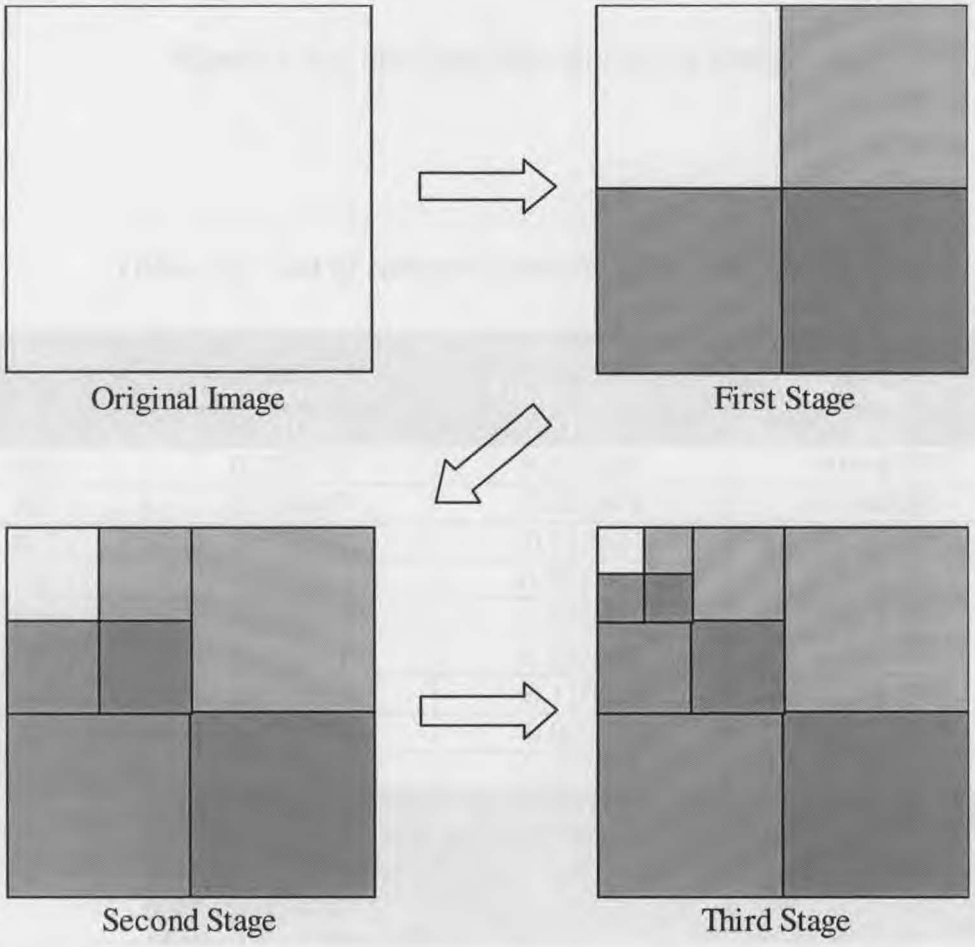


Figure 2-19: 2D DWT subband subdivision

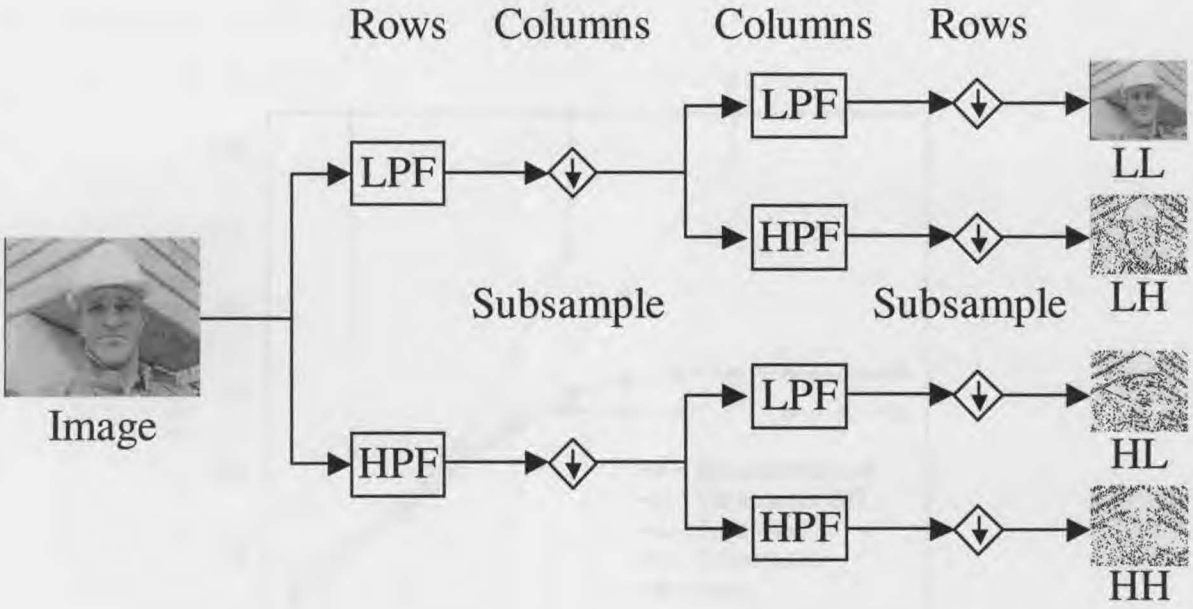


Figure 2-20: Subband filtering on an image

Table 2-3: List of common wavelet filter coefficients

Coefficients	Non-Integer Wavelets				
	Daubechie's-8 (H_0)	Villasenor-9/7 (H_0)	Villasenor-9/7 (G_0)		
W(0)	0.230378	0.037828	-0.064539		
W(1)	0.714847	-0.023849	-0.040689		
W(2)	0.630881	-0.110624	0.418092		
W(3)	-0.027984	0.377402	0.788486		
W(4)	-0.187035	0.852699	0.418092		
W(5)	0.030841	0.377402	-0.040689		
W(6)	0.032883	-0.110624	-0.064539		
W(7)	-0.010597	-0.023849			
W(8)		0.037828			
	Binary (Integer) Wavelets				
	Two-Six/√2 (H_0)	Two-Six/√2 (G_0)	Triangular (H_0)	Triangular (G_0)	Haar * √2 (H_0)
W(0)	1/2	-1/16	0	1/2	1
W(1)	1/2	1/16	1	1	1
W(2)		½	0	1/2	
W(3)		½			
W(4)		1/16			
W(5)		-1/16			

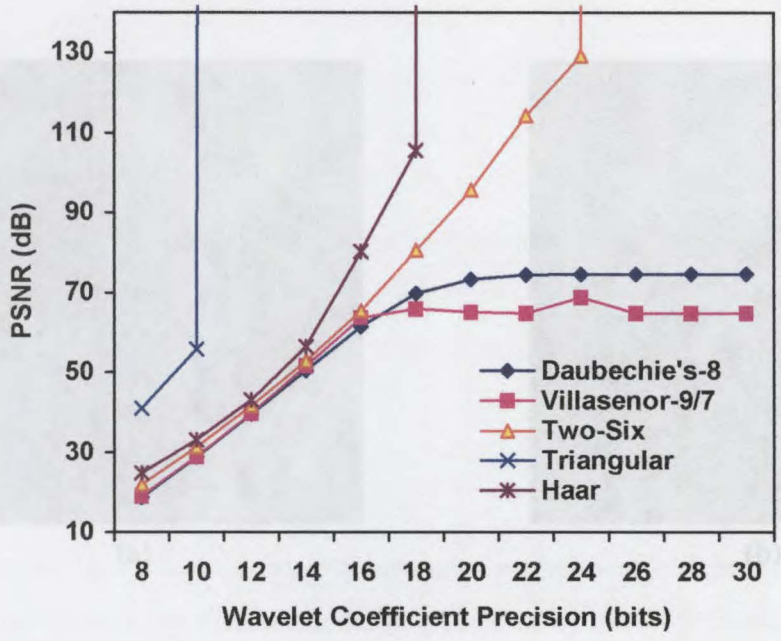


Figure 2-21: Effect of precision on filter performance

Figure 2-21 is a depiction of the performance of some of the more common DWTs where the register precision has been limited in the number of bits. The quick convergence to near perfection of the binary filters (namely Triangular and Harr) should be noted.

2.5.2. Triangular Binary Wavelet

Typically, realising a DWT requires the application of a convolution between the signal and the filter coefficients. For larger multi-tap, non-integer filters such as Villasenor the computational complexity generally limits the method primarily to software. The research performed in [44] suggests that the triangular binary wavelet is not only simple in nature for hardware implementation but also performs sufficiently as a transform filter for the DWT.

Figure 2-22 illustrates the 'Lena' image quantised to 0.2 bits per pixel using (a.) Villasenor filters and (b.) triangular filters. As is apparent, the variance in visual

appearance quality of the images is very subtle with the Villasenor filters accommodating slightly higher frequency components.

(a) (b)

Figure 2-22: 'Lena' image @ 0.2 bpp

(a) Villasenor

(b) Triangular

The simplicity, in terms of implementation, of the triangular wavelet transform is attributed to three key factors;

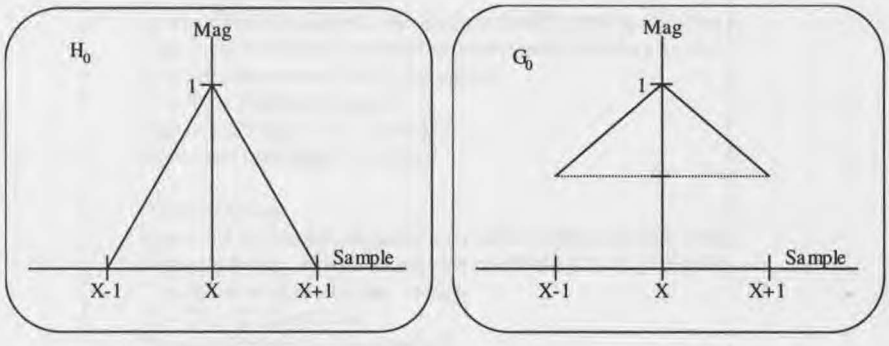


Figure 2-23: High & low pass triangular filter coefficients

1. **The low-pass coefficients are directly copied.** Figure 2-23 indicates that the low-pass coefficients are generated from the $H_0 = [0\ 1\ 0]$ filter. Since the only multiplier is a factor of one, the generated coefficients are literally a sub-sampled version of the original coefficients (or image values). Therefore the

low-pass component constitutes of each alternate coefficient from the original set.

2. **High-pass coefficients only require shifts and summations.** Figure 2-23 also shows that the high-pass filter, $G_0 = [0.5\ 1\ 0.5]$, consist of two divisions and two summations. The 0.5-filter coefficients are implemented by simple binary right-shifts of the image coefficients, while the 1-filter coefficients are realised by a simple copy. The resultant G_0 is attained by summing these three components together.

3. **2D performed via multiple 1D transforms.** Due to its orthogonal nature, the 2D transform is easily realised by performing several single dimensional transforms on the rows and columns of the image independently. These 1D transforms can be conducted in parallel in a single direction (either rows or columns) at a time.

The algorithm for the forward and inverse 2D triangular DWT can be seen in Figure 2-24 and Figure 2-25, respectively.

```
For S = 1 to scales
  X = select all low pass coefficients in scale

  First perform on columns
  Low = all odd index valued column coefficients in all rows
  High = all even index valued column coefficient in all rows
  T1 = left column shift all Low values
  T1 = T1 + T1(Last Column)
  High = ((2*High) – T1 – Low)/2
  Store Low and High in output

  Perform Rows
  Low = all odd index valued row coefficients in all columns
  High = all even index valued row coefficient in all columns
  T1 = left row shift all Low values
  T1 = T1 + T1(Last Row)
  High = ((2*High) – T1 – Low)/2
  Store Low and High in output

End for
```

Figure 2-24: Forward triangular DWT algorithm

```
For S = 1 to scales
  X = select all low pass coefficients in scale

  First perform on columns
  Low = all odd index valued column coefficients in all rows
  High = all even index valued column coefficient in all rows
  T1 = left column shift all Low values
  T1 = T1 + T1(Last Column)
  High = ((2*High) + T1 + Low)/2
  Store Low and High in output

  Perform Rows
  Low = all odd index valued row coefficients in all columns
  High = all even index valued row coefficient in all columns
  T1 = left row shift all Low values
  T1 = T1 + T1(Last Row)
  High = ((2*High) + T1 + Low)/2
  Store Low and High in output

End for
```

Figure 2-25: Inverse triangular DWT algorithm

2.5.3. Pyramidal vs. Nucleic Wavelet Blocks

Coefficients generated as when evaluating a DWT are typically organised in one of two representation formats; pyramidal as in [22] or Nucleic block based as in [23] [24]. Figure 2-26 contrasts the difference between the two, for a 3-scale transform conducted on an 8 x 8 block. As can be observed the positioning of the resultant coefficients varies quite significantly.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

1	5	3	6	2	7	4	8
33	37	34	38	35	39	36	40
17	13	19	14	18	15	20	16
41	45	42	46	43	47	44	48
9	21	11	22	10	23	12	24
49	53	50	54	51	55	52	56
25	29	27	30	26	31	28	32
57	61	58	62	59	63	60	64

Figure 2-26: Pyramidal vs. Nucleic coefficient arrangement

The pyramidal format stems mainly from literature and more traditional techniques employed to perform the DWT [41]. It arranges the coefficients in a manner that is

easy to view and access, with related coefficients being scattered throughout the block or coefficient map. The nucleic scheme originates as a by-product of performing the DWT in a parallel array based manner, and its main advantage is that all related coefficients belonging to a particular set of spatial frequencies are grouped together in a single nucleic block, whereas the pyramidal scheme can distribute these contents across the entire image coefficient map. Therefore, the nucleic block based approach is more appealing for hardware implementation where long paths present problems.

Figure 2-27 represents a 16 x 16 pixel image block, transformed using three-scales and arranged into the pyramidal format. As can be obviously noted, a large number of related coefficients in different subbands span nearly half the width of the array. In comparison Figure 2-28 represents a 16 x 16 block illustrating the nucleic block scheme. A significant number of connections remain quite short in distance, however, a few connections can travel the maximum distance of 2^{N-1} pixels, where N represents the number of scales. The example connections provided in the two figures illustrate the savings in connection distance that can occur if nucleic block are employed.

2.6. Quantisation (Stage C)

The coefficient quantisation stage introduces the primary lossy component in current standard image/video compression codecs. The pre-calculated loss in number of bits used to represent each of the coefficients has resulted in the term 'lossy compression' being identified with this stage. Hence, coefficients that have been quantised generally approach the value of zero or are represented in a fewer scale range gradients. Considering an analogue range represented by 8-bit coefficient values, it is easily identified that representing these values with 6-bit values reduces the number of range divisions from 256 to 64. Therefore representing the range 0-255 by 6 bits of precision results in the coefficients being grouped into multiples of 4 levels (eg.0..4..8..12.....255), taken from the original step size. A typical quantisation equation, especially in relation to hardware implementation, can be seen in (Eq. 2.10). Where C is the original coefficient, C' is the quantised coefficient, N, the original number of bits used to represent the coefficient and M, the new number of bits used to represent the coefficient.

$$C' = floor\left(C \times 2^{(M-N)}\right)$$

(Eq. 2.10)

1	1	2	2	3	4	3	4	5	6	7	8	5	6	7	8
1	1	2	2	11	12	11	12	13	14	15	16	13	14	15	16
9	9	10	10	3	4	3	4	21	22	23	24	21	22	23	24
9	9	10	10	11	12	11	12	29	30	31	32	29	30	31	32
17	18	17	18	19	20	19	20	5	6	7	8	5	6	7	8
25	26	25	26	27	28	27	28	13	14	15	16	13	14	15	16
17	18	17	18	19	20	19	20	21	22	23	24	21	22	23	24
25	26	25	26	27	28	27	28	29	30	31	32	29	30	31	32
33	34	35	36	33	34	35	36	37	38	39	40	37	38	39	40
41	42	43	44	41	42	43	44	45	46	47	48	45	46	47	48
49	50	51	52	49	50	51	52	53	54	55	56	53	54	55	56
57	58	59	60	57	58	59	60	61	62	63	64	61	62	63	64
33	34	35	36	33	34	35	36	37	38	39	40	37	38	39	40
41	42	43	44	41	42	43	44	45	46	47	48	45	46	47	48
49	50	51	52	49	50	51	52	53	54	55	56	53	54	55	56
57	58	59	60	57	58	59	60	61	62	63	64	61	62	63	64

Figure 2-27: Related coefficient distance in pyramidal scheme

1	5	3	6	2	7	4	8	1	5	3	6	2	7	4	8
33	37	34	38	35	39	36	40	33	37	34	38	35	39	36	40
17	13	19	14	18	15	20	16	17	13	19	14	18	15	20	16
41	45	42	46	43	47	44	48	41	45	42	46	43	47	44	48
9	21	11	22	10	23	12	24	9	21	11	22	10	23	12	24
49	53	50	54	51	55	52	56	49	53	50	54	51	55	52	56
25	29	27	30	26	31	28	32	25	29	27	30	26	31	28	32
57	61	58	62	59	63	60	64	57	61	58	62	59	63	60	64
1	5	3	6	2	7	4	8	1	5	3	6	2	7	4	8
33	37	34	38	35	39	36	40	33	37	34	38	35	39	36	40
17	13	19	14	18	15	20	16	17	13	19	14	18	15	20	16
41	45	42	46	43	47	44	48	41	45	42	46	43	47	44	48
9	21	11	22	10	23	12	24	9	21	11	22	10	23	12	24
49	53	50	54	51	55	52	56	49	53	50	54	51	55	52	56
25	29	27	30	26	31	28	32	25	29	27	30	26	31	28	32
57	61	58	62	59	63	60	64	57	61	58	62	59	63	60	64

Figure 2-28: Related coefficient distance in nucleic scheme

Since performing uniform quantisation over all coefficients doesn't necessarily represent an efficient quantising codec, three key factors influence the decision;

1. **Coefficient magnitudes resulting from a particular transform** – Typically the quantisation magnitude, in terms of its coefficients, is proportional to the average magnitude of the transform coefficients. For instance, since magnitudes resulting from an 8x8 DCT can increase up to 64 times the original pixel values, a much larger quantisation coefficient is applied when compared to wavelets which only double in value.

2. **Coefficient significance, in reference to its corresponding spatial frequency** – Depending on the delocalisation algorithm employed to perform the transform, varying quantisation levels are applied to coefficients representing different spatial frequencies. Typically, coefficients representing less significant spatial frequencies are quantised more than those of more significance. For instance the DC coefficient in a DCT is quantised relatively less in comparison to the rest. Figure 2-29 represent the pattern of a typical DCT significance coefficient map with the more significant coefficients located closer to the top left corner (brighter) and less significant coefficients (darker) arranged closed to the bottom right. The brighter components are quantised less in comparison to the darker components typically following a psycho-visual quantisation matrix.

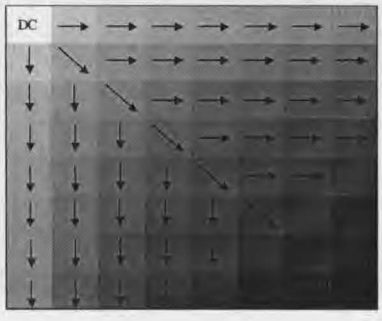


Figure 2-29: DCT Significance Map

3. **The accommodation of a particular quantisation scheme in a selected algorithm** – Commonly the quantisation scheme chosen is designed to fully integrate with encoding algorithm as it directly influences the final bit-count for a particular image or transform. Inadequacies in the quantisation scheme generally result in the reduction of features or efficiency of the coder. For

instance, the quantisation scheme typically used with a DCT is not necessarily optimal for a wavelet-based coder. In addition features such as multi-resolution and scalable video, which is more suited to wavelet based systems due to its subband hierarchy, require special considerations like successive approximation to improve its efficiency.

2.6.1. DCT Quantisation

The standards H261, H263, MPEG1 2 & 4, which employ some form of the DCT in the base video coding algorithm, use predetermined tables of quantisation values, for the coefficients resulting from the DCT. Since the block size of a DCT is fixed at 8 x 8 coefficients, a fully researched quantisation value-set from a selected list is applied individually to all the coefficients, and an index to the list is supplied to the decoder to enable correct reconstruction.

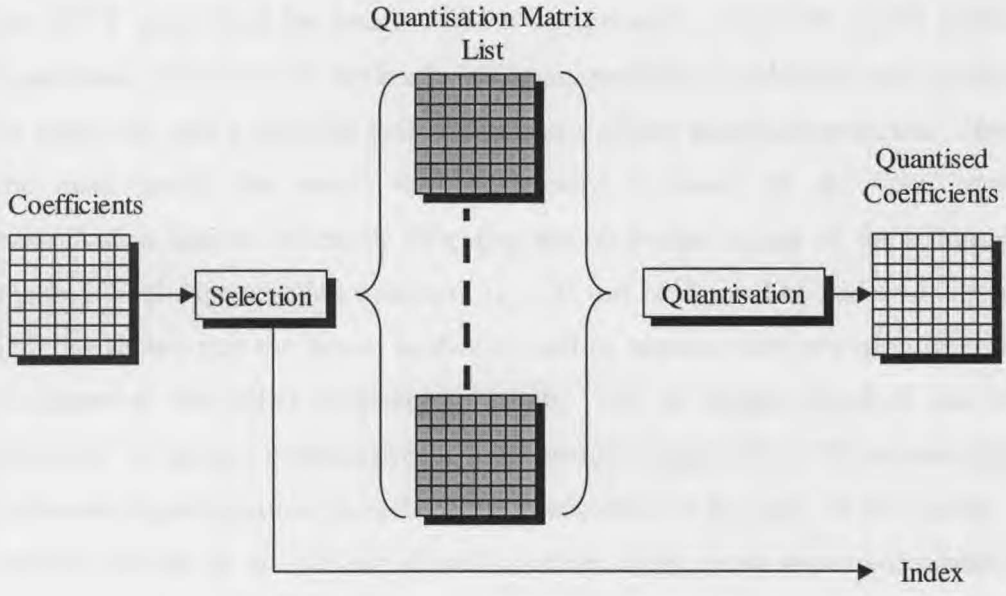


Figure 2-30: DCT Quantisation

Figure 2-30 illustrates the quantisation process as carried out for each 8 x 8 coefficient block. Although the standards do not dictate the selection system it is generally based on the average energy of each block. Recent algorithms such as MPEG 2 and 4 also allow for the inclusion of a user defined quantisation matrix if required.

2.6.2. Subband Quantisation

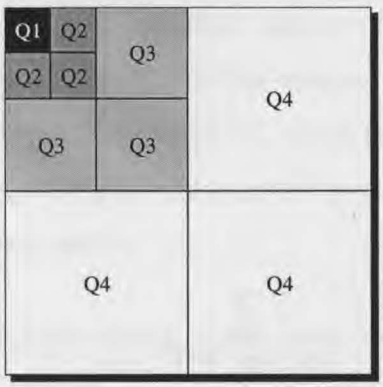


Figure 2-31: Subband Quantisation

Subband quantisation is a relatively newer method when compared to the more traditional uniform / non-uniform quantisation techniques applied on static sized frequency blocks [22]. Its existence is generally attributed to the advent of wavelet based image decomposition techniques, and as such it is generally closely coupled with the DWT when used for image / video compression. Since the DWT performs a logarithmic subdivision of the image into multi-resolution subbands, each subband can be quantised with a different uniform or non-uniform quantisation matrix. However, the main hurdle that needs to be overcome is based on the development of prioritisation scheme efficiently allocating the bit budget across all the subbands. In Figure 2-31 the quantisation matrices Q1 – Q4 can be derived on order of importance. [35] has shown that the lowest subbands tend to require more precision in general as compared to the higher frequency subbands. The bit budget therefore, can be thus allocated. A further, more analytical, improvement suggested in [22] involves applying successive approximation to each of the coefficients in the each of the bands. This not only allows for an efficient allocation of bits to the more important subbands but also allows for features such as progressive video coding and precise rate control. Chapter 3 tackles the concepts of successive approximation more thoroughly.

2.7. Reconstructed Image Quality

Since quantisation generally degrades the quality of the reconstructed or decoded image, it becomes vital that this be quantitatively or qualitatively expressed. This aids in the comparison of different compression systems and their effect on different image types.

2.7.1. Peak Signal to Noise Ratio (PSNR)

The PSNR has become the adopted standard measure employed to compare image quality between different quantisation and coding systems. The PSNR is a quantitative measure and is calculated by evaluating (Eq. 2.11), which takes into account the original and the reconstructed image. It is generally accepted in the field that a PSNR of 25dB or higher is of acceptable image quality.

$$PSNR = 10\log\left(\frac{\sum I^2}{\sum (R-O)^2}\right) db$$

(Eq. 2.11)

Where, I represents an N x M block of peak signal values (i.e. $2^{\text{max number of bits}} - 1$), R is an N x M block for the reconstructed image, and O is an N x M block for the original image. N x M represent the number of pixels in an image.

PSNR is a computational method for comparing image quality, however simple scaling factors in the reconstructed image which can be deemed acceptable after visual appraisal, easily defeat this PSNR test.

2.7.2. Visual Quality

Even though the PSNR measure is a reasonable estimate, a full visual appraisal is also often performed on reconstructed images. Artefacts such as image scaling easily mislead the PSNR, while blocking artefacts which affect the PSNR less may be considered visually unacceptable. Figure 2-32 and Figure 2-33 both show images quantised to a PSNR of 30 dB. However one might consider Figure 2-33 to be more visually pleasing.

Figure 2-32: 0.21 bpp Image with 30 dB PSNR

Figure 2-33: 0.19 bpp Image with 30 dB PSNR

2.8. Entropy Coding (Stage D)

Section 2.6 introduced a lossy component in the general image compression algorithm. The lossy component typically reduces the number of bits required to represent a coefficient or a set of coefficients. However, it does not provide a mechanism for the elimination of redundant data usually found inherent to the images or video, for instance large areas that are the same in colour or texture typically produce a set of coefficients with a few larger in value and the majority very close to zero. Entropy coding is based on the loss-less coding of these large expanses of similar valued coefficients. Generally the more efficient codecs (MPEG 4, Wavelet Based codecs etc.) tend to employ an adaptive method to identify and code redundant coefficient areas [42] [43]. Some entropy coding techniques are outlined below.

2.8.1. ZigZag & Run Length Coding

Zigzag and run-length coding [8] originate from a typical block-based image or video codec such as JPEG, H.261, H.263, MPEG 1 [12], MPEG 2, etc., which generally use the DCT transform and related quantisation. The process has two main purposes,

1. To convert a 2D block into a 1D vector of coefficients for ease of transmission
2. To minimise the redundancy of repeated coefficients within the vector.

Figure 2-34 is a typical zigzag-coding map employed to code a typical 8 x 8 DCT coefficient block. It exploits the key features of a DCT and associated quantisation mechanism, to encode coefficients where the high frequency coefficients are significantly quantised or zero. Therefore producing a coefficient list that contains large DC values at the beginning and smaller or zero coefficients towards the end. This enables the use of a simple redundancy coding technique to be applied to group of like high frequency components within the block together, i.e. run-length coding.

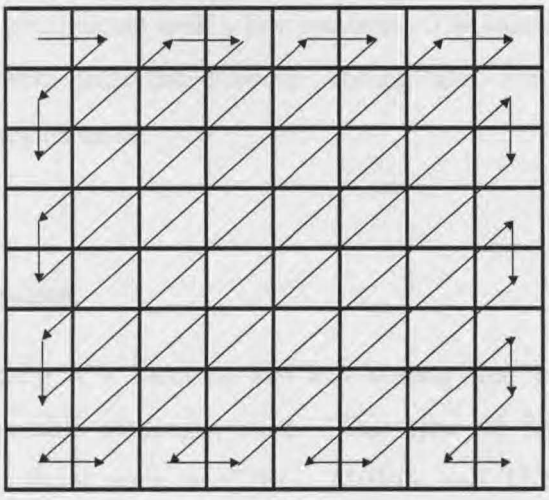


Figure 2-34: Zigzag Coding Technique

Run-length coding is a technique that generates a symbol to represent a larger run of one particular type of coefficient followed by another. For example in Figure 2-35 a different symbols can be allocated for the following runs.

SymbA = 2221

SymbB = 00002

SymbC = 00001

70	-40	38	10	2	2	2	1	0	0	0	0	2	0	0	0	0	1	...

Figure 2-35: Run Length Coding Example

The symbols employed are typically statistically chosen by investigation of a large volume of images, and are typically fixed for a particular standard [8] [11]. Finally these symbols can then be Huffman or arithmetic coded to eliminate nearly all redundancy.

2.8.2. Zerotree Coding Overview

This form of entropy coding is fully covered in Chapter 3, as this thesis is based on the implementation of such a codec within a specific parallel architecture. The zerotree coding technique is generally performed on full image transforms such as the Discrete Wavelet Transform, and as such exploits the self-similarity or relations between subbands resulting from a DWT. These relations are then used to efficiently code large

areas of insignificant coefficients with a few symbols. The quantisation schemes used also tie in very closely with the zerotree coding and are generally chosen to accommodate some key features.

2.8.3. VLC Coding

Variable Length Coding is a form of loss-less coding that codes symbols with a statistically chosen variable bit-length code. This type of final entropy coding is typically used in standards such as JPEG, MPEG and H.261/3, which have a statistically defined set of variable length codes to accommodate any variations in the image. VLC coding, especially in the mentioned situations, has two simple flaws

1. The minimum code generated is of integer bit-length (i.e. 1 bit per symbol)
2. Does not adapt to any image characteristics.

The latter is easily solved by applying an adaptive algorithm such as Adaptive Huffman Coding [46] [47] [48] which attempts to 'customise' the coding statistics to the image or sequence at hand. However, the complexity and memory requirements increase significantly.

The generation of one bit per symbol is a flaw inherent to VLC coding techniques and as such is not easily overcome. Typically the more complex yet efficient Arithmetic Coding technique is employed to overcome this problem.

2.8.4. Arithmetic Coding

Arithmetic coding is a statistical coding technique that attempts represent source data with minimal entropy [49]. It is not a table lookup based approach like VLC / Huffman coding, therefore it does not require each symbol to be represented by an integer number of bits. Also, more than one symbol can be represented in less than one bit. It is this feature that makes this a better alternative to variable length coding as it approaches the Shannon [50] entropy bound.

The algorithm is based on an infinite range of real values ranging from 0 to 1. An example best illustrates the operation of the coder. Taking a 4 symbol systems, with symbols A, B, C and EOT, if having probabilities of occurrence 0.4, 0.3 0.2 and 0.1 respectively, are transmitted in the following sequence ACB EOT, then Figure 2-36 describes the operation.

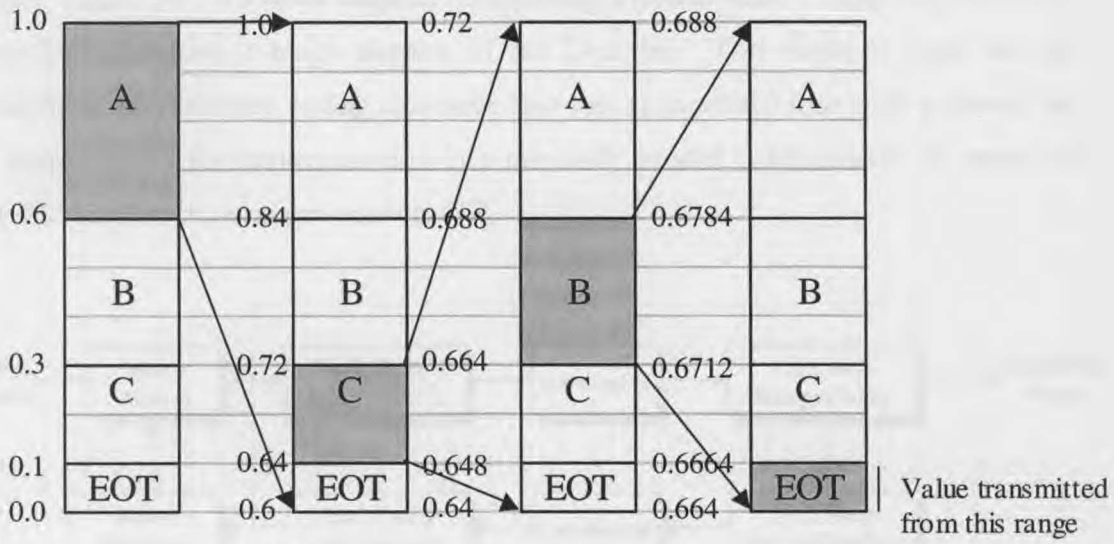


Figure 2-36: Arithmetic Coding Example

The first A limits the range between 0.6 and 1.0, then each of the other symbols alter the range according to (Eq. 2.12)

$$Low_N = Low_O + Range_O * Q_1$$

$$High_N = Low_O + Range_O * (Q_1 + P_N)$$

(Eq. 2.12)

Where Q_1 is the lower accumulated probability and P_n is the newer probability. Finally a value between 0.664 and 0.6664 is chosen and sent.

The decoder initially starts off with 0 and 1 as the range, then, since the number received is greater than 0.6 it knows the first symbol is an A, it then mimics the encoder until an EOT is extracted. This signifies the end of the symbol stream.

2.9. Image / Video Coding System

With these basic blocks in mind a general video / image compression codec can be realised. Figure 2-37 is a block diagram representing a typical video / image encoder while Figure 2-38 illustrates a block diagram of the Decoder. This thesis is based on the development of a zerotree coding technique that can be modelled into such a system but with consideration for implementation in a massively parallel environment. A survey of some VLSI architectures is presented in [72].

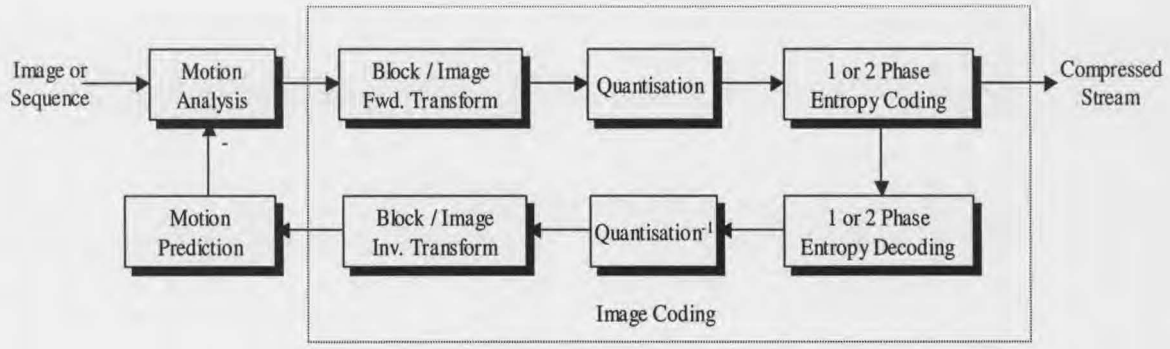


Figure 2-37: Image / Video Encoding

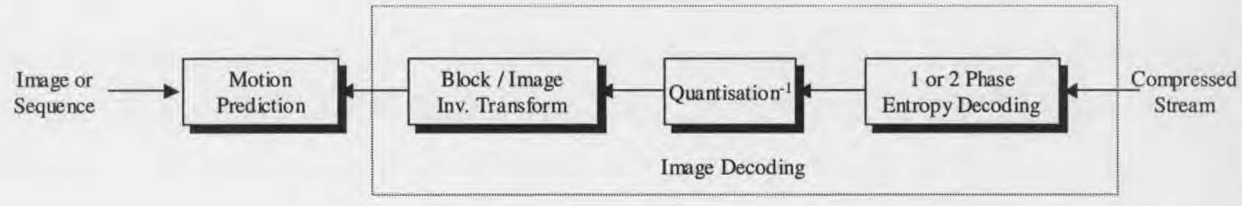


Figure 2-38: Image / Video Decoding

2.10. Conclusion

This chapter has presented a number of fundamental principals surrounding modern image and video compression methods. The concepts introduced include explanations of images, video, image sampling, image/video compression, test images used and codec components such as, motion analysis and compensation (Block Matching

Motion Estimation, frame differencing), image transforms (wavelets, Discrete Cosine Transform), image quantisation (subband) image/video coding schemes (zigzag, run-length) and entropy coding (Variable Length Coding, arithmetic). This chapter is intended as an introduction and a base for the chapters to follow.

Chapter 3

ZEROTREE CODING

"My designation is Seven of Nine, Tertiary Adjunct of Unimatrix Zero-One, but you may call me Seven of Nine."

Borg drone 7 of 9 introducing herself, *Star Trek: Voyager*

3.1. Introduction

In 1993 Jerome M. Shapiro [22] introduced a simple, yet remarkably effective image-coding algorithm based on a wavelet transform. He combined the wavelet transform with a novel representation of the transformed coefficients and termed this scheme zerotree coding. Zerotree coding, analogous to the Borg drone's request, can represent a larger piece of information in a more succinct manner. This hierarchical technique provided a means to represent the coefficients in terms of their magnitude, position and significance. The Embedded Zerotree Wavelet (EZW) coding technique exploited the hierarchical nature of the wavelet transform to categorise coefficients of similar spatial position, but at different frequencies, into trees that were based on the significance of the coefficients. Since its original acceptance many variations of this technique have been applied to image and video coding schemes alike, each providing some form of improvement over the standard coder [51] [52] [53]. In 1996 S. A. Martucci and I. Sodagar [54] also introduced a version of the zerotree coder based on a modified set of symbols that provided more uniform results compared to the EZW coder. It was identified as being more suitable for constant bit-rate video coding particularly for low bit-rate channels [55]. This technique termed ZeroTree Entropy (ZTE) coding became the standard adopted for the texture compression

subsystem in MPEG4. This thesis reports the work carried out to implement a zerotree coder in a massively parallel architecture to suit the Intelligent Pixel paradigm described in Chapter 4

3.2. The EZW Algorithm

The effectiveness of the EZW algorithm is directly influenced by the mechanism used to generate the coefficients, and as such the DWT has become the proven choice. This is because the spatial image content is arranged in the form of a hierarchical tree, which is a crucial requirement for the EZW. The DWT subdivides the spatial image content into time-frequency subbands that are easily hierarchically arranged. As a result, image components that contain higher spatial change are represented in the higher frequency subbands (HL1 — HH3, See Figure 3-1 and Figure 3-2) while the "smooth" changing areas are represented at the lower frequency bands (LL). The number of scales determines the number of segmentation bands that subdivide the spatial change frequency range (0 - f) in an image. To illustrate this refer to Figure 3-1 and Figure 3-2. Figure 3-1 contains typical DWT subband decomposition map and a test image attributing sharp high contrast horizontal and vertical lines and an area of gradual spatial change in the middle. Figure 3-2 is the 3-scale DWT subband decomposition of the test image using a triangular wavelet filter. An obvious characteristic can be identified with the LL band, in that it resembles a sub-sampled version of the original image. The other bands fall into one of three categories, the horizontal frequency meta-tree, the vertical frequency meta-tree and the diagonal frequency meta-tree. The three trees, depending on the subband, show varying degrees of frequency components caused by the high contrast changes in the image. It is easily noticed that the gradual changing components, the box in the middle, is almost fully represented in the LL band. However, the most important observation made, in terms of the EZW algorithm, is that the content of all the subbands are in some way related to a component in the LL band. This implies that coefficients in different bands are possibly related to each other via a tree structure.

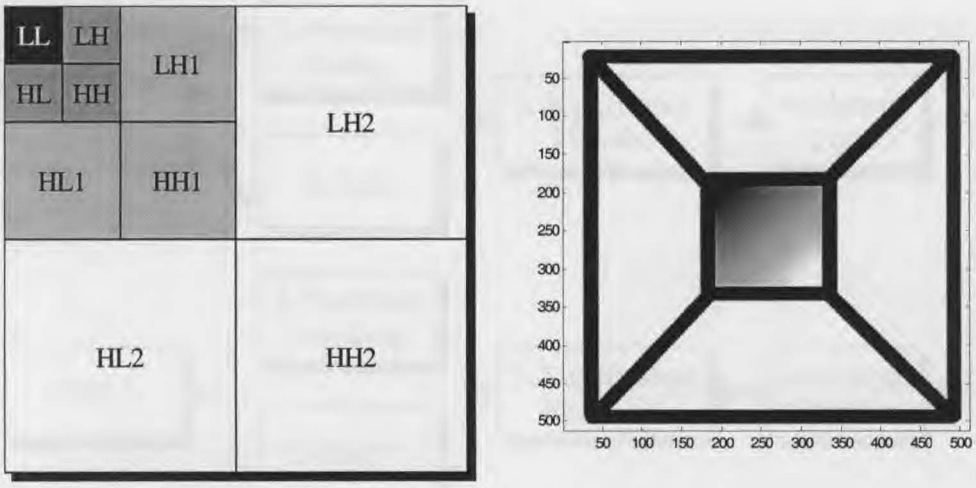


Figure 3-1: Test Image

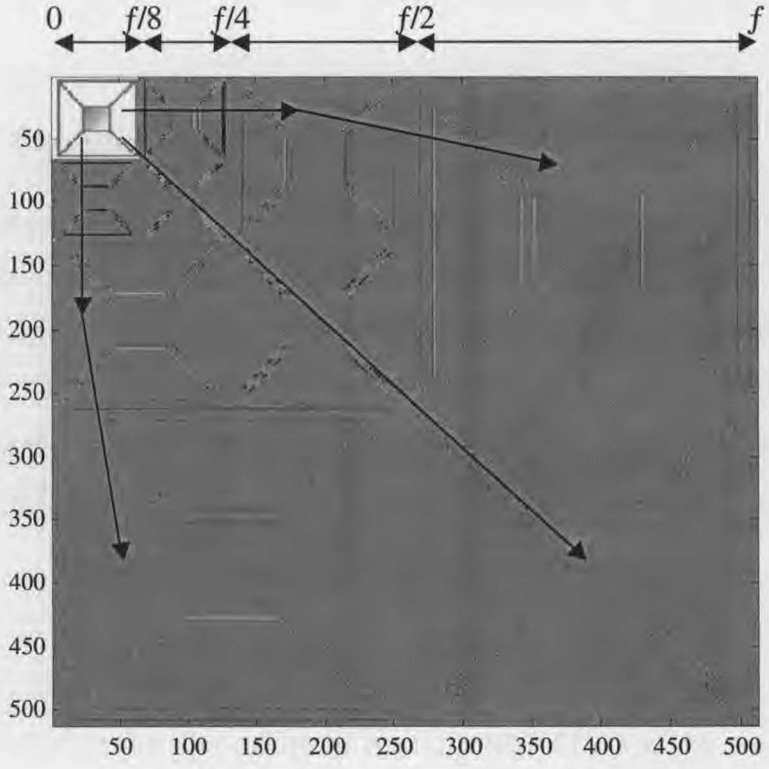


Figure 3-2: 3 Scale DWT of Test Image

The novelty, in terms of coding, exhibited by the EZW can be attributed to three major coding mechanisms within the algorithm. Figure 3-3 is a block diagram of the complete EZW coding-decoding process highlighting the main components

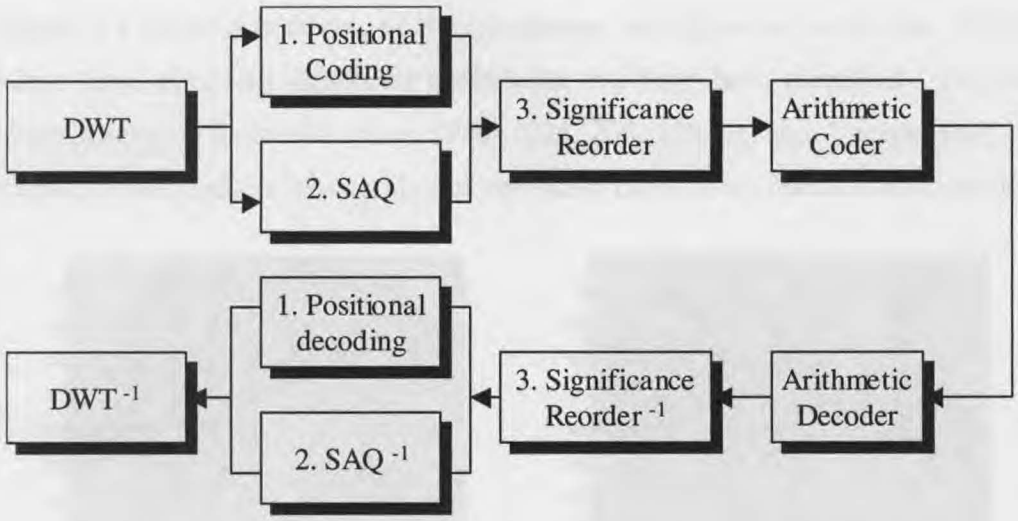


Figure 3-3: Main EZW Components (Encoder & Decoder)

3.2.1. Significance of Coefficients

Given a particular bit budget the EZW algorithm always attempts to represent an input image to as high a standard as achievable within that bit constraint. To accomplish this, a search to locate the more significant coefficients within the transformed image is performed. The significance of a coefficient is determined by performing $C \geq T$, where C is the coefficient and T is the current threshold value that is initialised with the result of (Eq. 3.1)

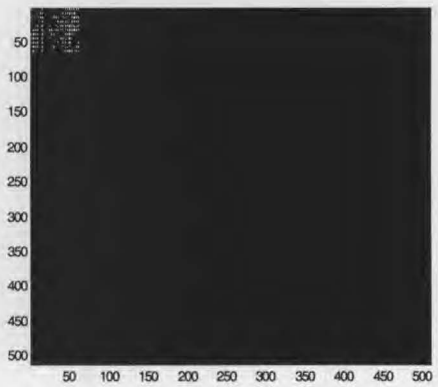
$$T = 2^{\lfloor \log_2 (MAX(|C(x,y)|)) \rfloor}$$

(Eq. 3.1)

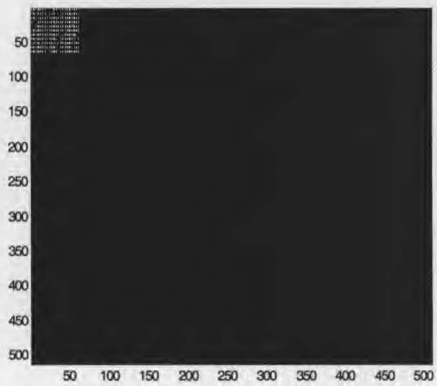
Where $MAX(x)$ is the max values of matrix x and $C(x,y)$ is the set of coefficients resulting from performing a DWT on an image.

Since the initial threshold value only identifies the most significant coefficients, and since these coefficients only represent a small subset of the DWT coefficients, the threshold is readjusted and reapplied in another iteration to identify a greater number of significant coefficients. This significance identification process is iterated until the threshold reaches a value that generates too many coefficients or is zero. With each subsequent iteration the threshold value is halved, i.e. $T_{n+1} = T_N / 2$. The 6 images in

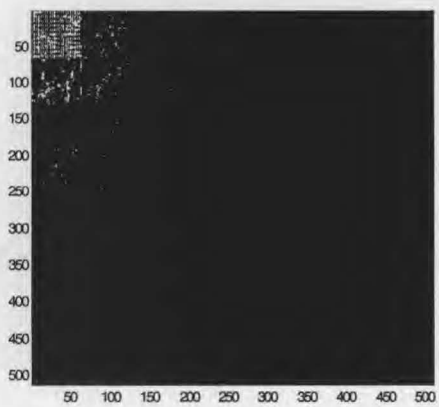
Figure 3-4 shows 6 iterations of the significance identification mechanism, where the white 'dots' represent significant coefficients that have been identified. The images show results of threshold values 2048, 1024, 256, 128, 16 and 1 respectively. The values in italics indicate the number of significant coefficients found in that iteration.



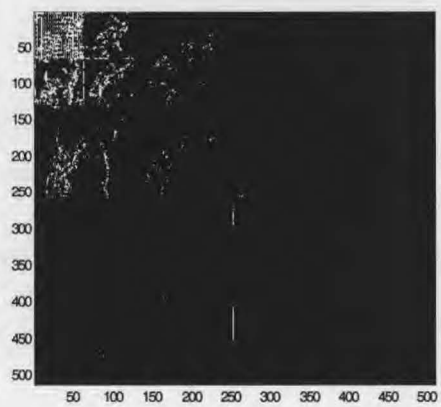
(a) (*476*)



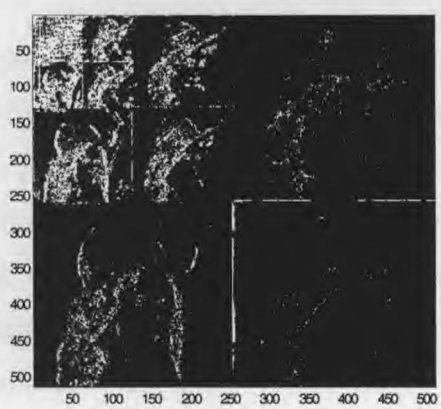
(b) (*905*)



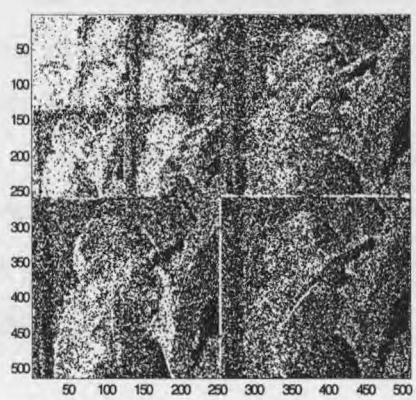
(c) (*1050*)



(d) (*2169*)



(e) (*19247*)



(f) (*142362*)

Figure 3-4: Significance Iterations

Generally lower threshold values identify a larger number of significant coefficients, which implies that a suitable compromise is often needed. Figure 3-5 illustrates this trend for the coefficient image used for Figure 3-4. The cumulative total indicates the total number of significant coefficients detected.

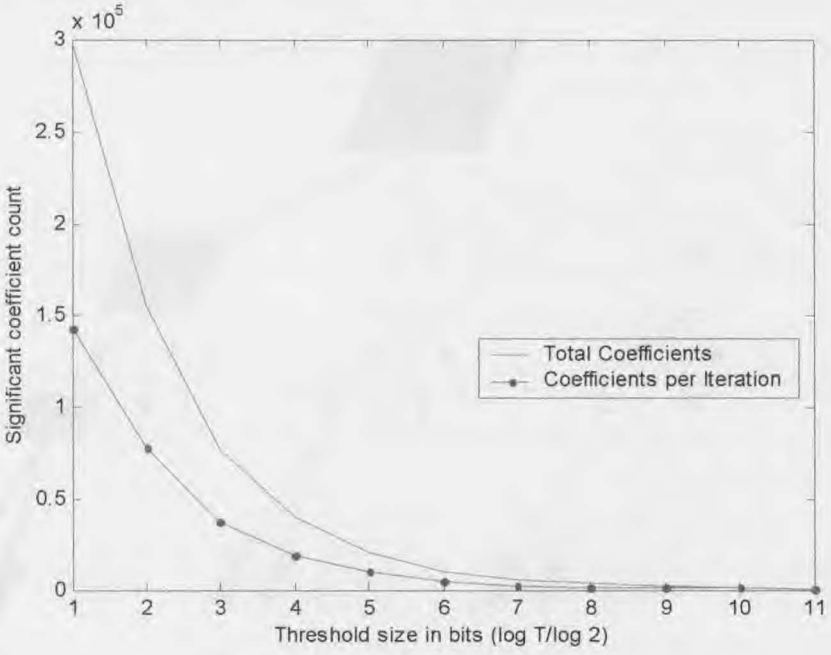


Figure 3-5: Significant coefficient count for varying threshold values

3.2.2. Positional Coding of Coefficients

Previously it has been noted that the key factor that allows for effective zerotree coding stems from the DWT, where coefficients in all higher bands have relations to a coefficient in the LL band. For instance Figure 3-2 clearly shows the relation between the LL band and the HH3 band, which contains the diagonal high frequency components of the LL band, but at a greater resolution. The useful implication here is that, the LL band forms the top node of set of "relation trees" that spans the entire transformed image. Figure 3-6 shows a typical "relation tree" that results for each coefficient (or pixel) in the LL band.

An important implication of these "relation trees" ties in with the decaying spectrum [22] nature of coefficients typical of wavelet transforms, in that the existence of a

significant coefficient in a higher frequency band depends heavily on the existence of an equal or more significant coefficient in a related lower frequency band. This then, justifies the use of zerotrees, which imply that a coefficient in a high frequency band is likely to be zero if a related coefficient in the low frequency band is zero. In practice exceptions to this exist and are dealt with differently within the zerotree algorithm.

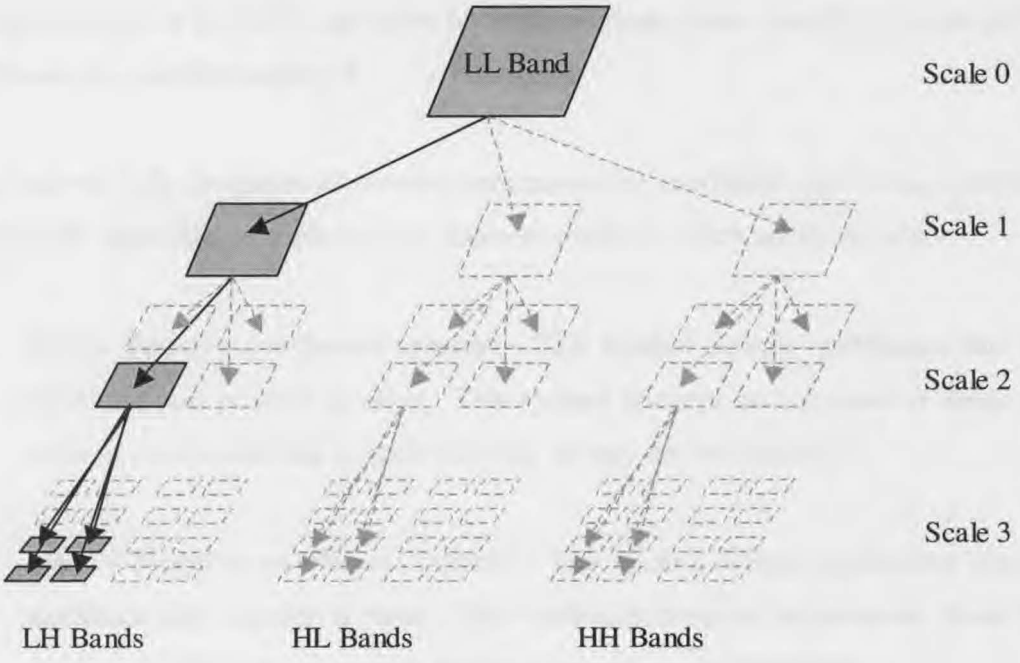


Figure 3-6: EZW Relations Trees

In general each parent node is related to four different sub-nodes (children) belonging to the immediate higher frequency subband. The only exceptions to this are the coefficients located in the LL band, which have only three sub-nodes, and coefficients located in the last scale-3 subband, which have no sub-nodes. With the exception of coefficients in the LL band, all of the remaining coefficients have exactly one parent node. Coefficients in the LL band have no parent nodes.

When encoding, the EZW algorithm takes advantage of these "relation trees" to describe the spatial coordinates of insignificant coefficients in high frequency subbands. A special symbol, termed a zerotree symbol, is allocated to describe a set of insignificant children coefficients that have a parent that is also insignificant. The advantage here is the ability to represent entire "trees" of insignificant coefficients with one symbol and still decode the positions of these insignificant coefficients correctly.

This is similar to the run-length mechanism employed in typical DCT compression systems, where the large lengths of zeros are represented by count values. However, the improvement in the EZW algorithm lies in its ability to define large sequences of zeros directly in two-dimensional space. A more generalised 2D hierarchical coefficient tree partitioning approach to image compression termed "Set Partitioning of Hierarchical Image Trees (SPHIT)" is investigated in [7]. This algorithm is an improvement on the EZW algorithm but is significantly more complex in terms of the symbol tree searches employed.

In order to fully categorise all possible occurrences of coefficient significance patterns, the EZW algorithm depends on four different symbols, which are listed below.

1. ***POS*** – **Positive coefficient symbol** – This symbol defines coefficients that are significant and positive in value. This symbol conveys no information about the children coefficients and as such they may or may not be significant.

2. ***NEG*** – **Negative coefficient symbol** – This symbol defines coefficients that are significant and negative in value. This symbol conveys no information about the children coefficients and as such they may or may not be significant.

3. ***ZTR*** – **Zerotree Root** – This symbol defines coefficients that are both insignificant and have children coefficients that are insignificant. The compression efficiency of the EZW algorithm is generally proportional to the existing number of such symbols.

4. ***IZO*** – **Isolated Zero** – This symbol defines coefficients that are insignificant yet contain significant children coefficients. These symbols are used to categorise trees that do not follow the decaying spectrum phenomenon, as well as define previously significant parents which less significant descendant.

Figure 3-7 pictures a flowchart representation of the symbol identification algorithm that is performed for each coefficient during an encode cycle. The decode process uses the flowchart in Figure 3-8 to identify the coefficients with respective symbols.

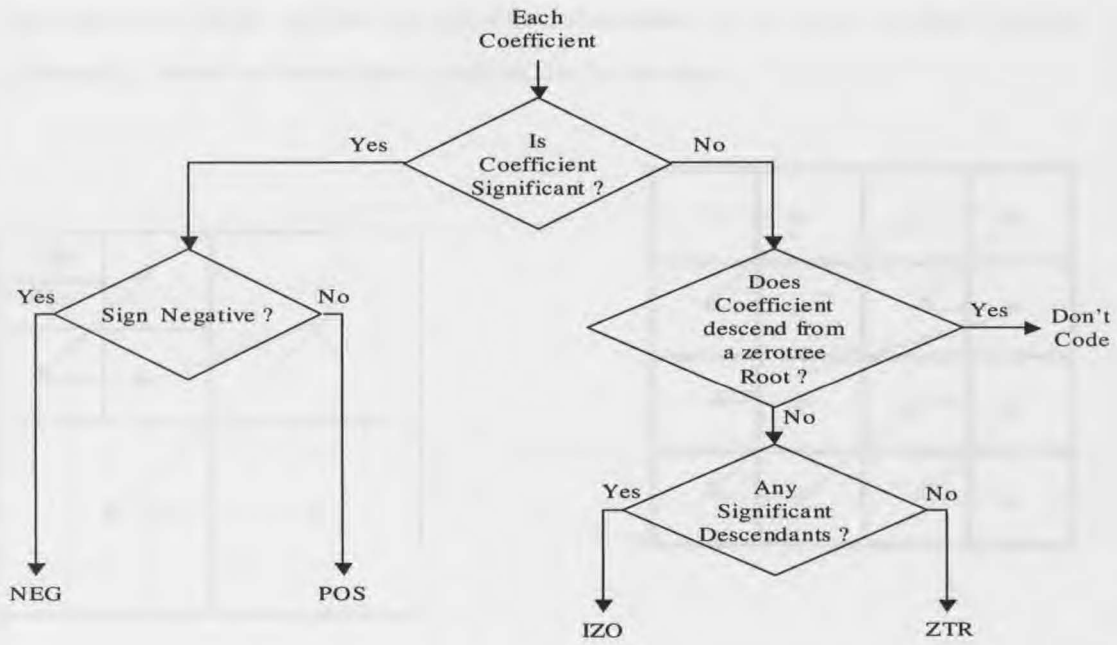


Figure 3-7: Symbol Identification Flowchart (Encode)

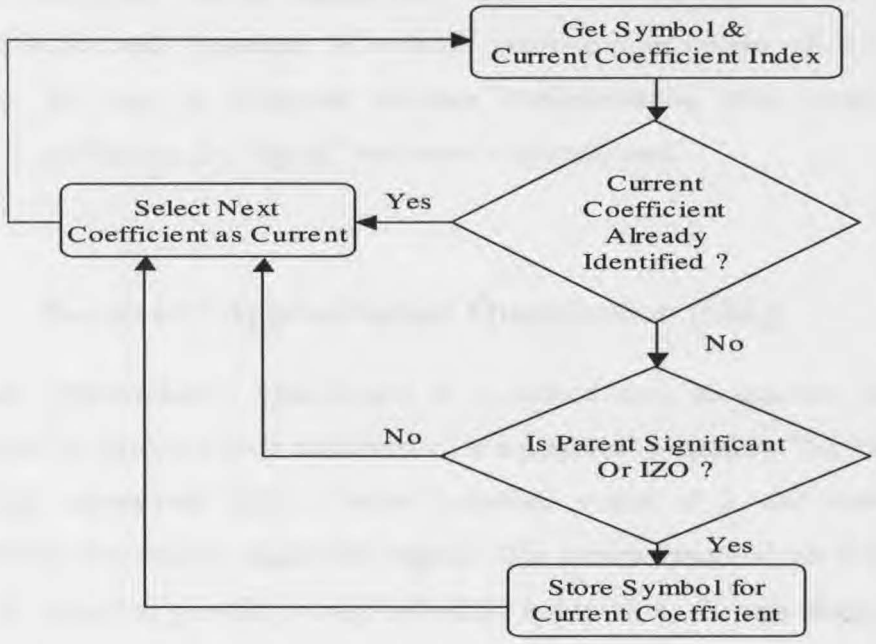


Figure 3-8: Coefficient Identification Flowchart (Decode)

Once the coefficient symbols have been identified, a special scanning technique is applied to order the symbols for correct transmission and decoding. Figure 3-9 illustrates the scanning order for firstly the subbands and then for coefficients within the subbands. This transmission pattern facilitates the correct decoding of large 2D

zerotrees, as a single symbol can provide information as to which symbols require information from the stream that is ordered in this manner.

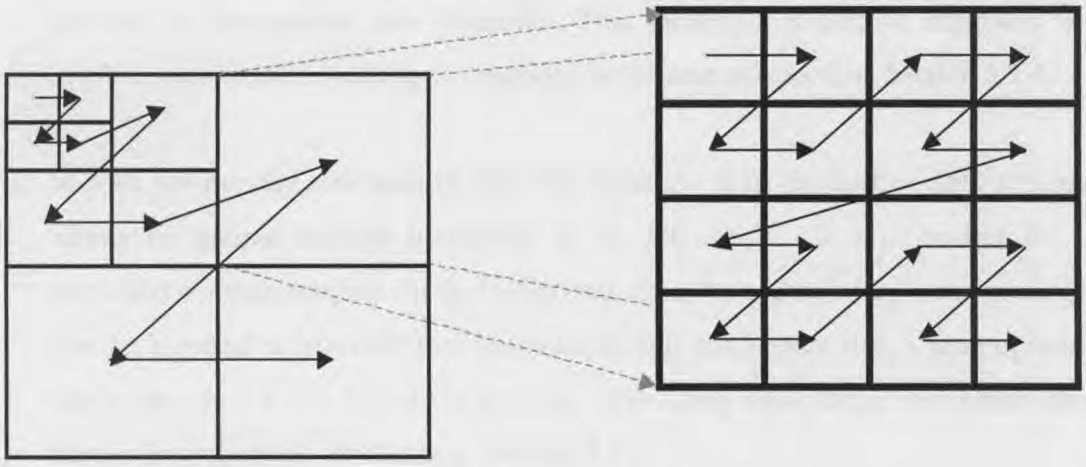


Figure 3-9: Scanning Order - Subbands & Coefficients

The strict compliance with the zigzag nature of the coefficient scanning within a single subband is not really necessary, as a simple raster-scan technique will also suffice. However, for ease of sequential software implementation when searching for significant coefficients, the "zigzag" technique is typically used.

3.2.3. Successive Approximation Quantisation (SAQ)

Successive Approximation Quantisation is a method used to quantise coefficient magnitudes (or introduce lossy compression) in a progressive manner. The coefficients are initially represented with a coarse quantised power of 2, and then refined progressively over time to equate the original. The incorporation of this scheme into the EZW algorithm provides a fully embedded approach to the quantisation of the coefficients, thus supplementing the positional coding mechanism with an efficient magnitude representation technique as well. Two reasons justify the use of such a technique over the flat quantisation model used in many typical compressing schemes such as JPEG, MPEG, and H261 etc.;

1. **Progressive image/video coding** – This is a technique whereby the detail levels of an image undergoing transmission are increased over time. This means that

decoding of an image can start before the full content of its equivalent compressed stream has been received. Therefore, at any one time the image decoded is a representation of the original image, but at a level of detail proportional to the amount of compressed data received. This technique is further improved by applying significance ordering as described in the next sub-section, Section 3.2.4.

2. **Simple yet precise rate control over the coding** – A by-product of SAQ scheme allows for simple bit-wise truncation of the bit-stream. If a bit-budget for a particular communications channel is defined, then the coding of a particular image can be stopped at precisely that bit-count in full confidence that a near optimal image decode for that bit-rate is possible. Providing significance reordering also further improves this mechanism. Section 3.2.4.

The SAQ mechanism is always initialised at the first significance identification phase (Section 3.2.1). This phase then defines the initial quantisation level. Once the significant coefficients have been identified more detail relating to these coefficients can be coded into the transmission stream. The operation is generally based on the segmentation of the coefficient plane (matrix) into bit-planes, which are then systematically ordered and transmitted.

Decoding can begin anytime after the first quantisation level for the image has been received, subsequent refinement data can then be used to improve the image quality.

Figure 3-10 is composed of a set of images representing a single image being progressively decoded under a SAQ scheme. A clear improvement in the image is seen (Top to bottom and left to right) as more refinement passes are made. If all the passes are transmitted then this scheme will incur no loss in image quality. For this case, however, the bit-count, when compared to an uncompressed image actually increases by at least a factor of $(B+2)/b$, where the largest coefficient can be represented in B bits and the largest image magnitude can be represented by b bits. This is an approximation as the effect of the positional coding scheme is highly dependant on the image selected. For the 'Lena' image this calculation indicates that the bit-count increases by a factor close to 1.87, which implies a size of approximately 3.9 Mbits, if all passes are performed

Figure 3-10: SAQ Refinement process.

The EZW algorithm simply employs a single binary (0, 1) digit to represent each coefficient refinement symbol. The symbols used are

$$0 - \text{For the coefficient } C < T.$$

$$1 - \text{For the coefficient } C \geq T.$$

where T represents the current threshold. Since only two types of symbols are used and since the probability of occurrence is heavily dependant on the image characteristics, performing arithmetic coding may have no effect, and as such the resulting bit-stream entropy may be increased [22].

3.2.4. Significance Reordering

A useful feature inherent to the EZW algorithm is its ability to provide a data bit-stream that can be easily truncated at any moment, yet represent an image as close to

the original as possible within a given bit-budget. This feature is mainly attributed to the SAQ process. However, by reordering the coefficients in a manner where the refinements for the most significant decoded coefficients are transmitted before those lesser in significance, a "best" representation of the original image is possible within the given bit-budget. The EZW algorithm incorporates this mechanism into its coding technique to fully enhance the positional coding and SAQ mechanisms. For example, if in a pass three coefficients were decoded, i.e. 43, 55 and 61, then during the refinement phase the order of the refinement bits arrive in reverse, as the coefficient 61 receives the first refinement bit, followed by the coefficient 55 and finally 43. In this manner if the bit-stream was truncated mid-stream of the refinement pass, the most significant coefficients, which have a greater bearing on the decoded image, receive the most refinement bits. Although this is a very useful technique it proposes rather significant hurdle in terms of hardware implementation, as the coefficient list has to be constantly re-sorted. This feature is typically omitted in most hardware only EZW codecs due to the complex sorting structures required.

Multi-resolution image/video coding – This is a technique which facilitates the transmission of a single coded image or video sequence to any number of possible targets with varying bit-budgets relating to the available bandwidth constraints. For instance a single high quality image coded for a T1 internet connection can have its bit-stream truncated to suit a modem. A lower resolution image will be generated at the modem end, yet it maintains the "best" possible representation of the original image given this bit budget.

3.2.5. EZW Encoding / Decoding Process

An example illustrating the EZW encoding and decoding algorithm, as extracted from [22], is now presented to aid in the explanation. Only the preliminary passes are performed for this illustration. The stages within the algorithm are described below.

1. **Perform Wavelet Transform** – An example coefficient map resulting from a 3-scale DWT conducted on an 8 x 8 image is depicted in Figure 3-11. This example has an anomalous high frequency coefficient with a value of 47, which is relatively high when compared to its parent coefficients. The maximum magnitude is 63.

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
14	15	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 3-11: Example of an 8 x 8, 3-scale wavelet transform

2. **Set Initial Threshold Value** – The largest coefficient magnitude is 63, therefore it will require at least 6 bits to be fully represented. An initial threshold value of $2^{(6-1)} = 32$ is chosen as any power greater than 5 results in a threshold too large for the coefficients.

3. **Identify Significant Coefficients** – A search is then performed to identify all significant coefficients by comparing them to the threshold. All coefficients conforming to $C \geq T$, where $T = 32$, are now identified as significant.

4. **Generate Coefficient Symbols (Dominant Pass)** – The coefficient values 63, -34, 49 and 47 which have magnitude values greater than the threshold, generate the symbols ***POS***, ***NEG***, ***POS*** & ***POS*** respectively. Table 3-1 lists the set of symbols generated for the first significant pass for the entire array.

   The coefficient –31 in the LL and coefficient 14 in the HL band are both clearly insignificant. However, since coefficient 14 has a significant descendant, an ***IZO*** symbol is generated, and since coefficient –31 has an ***IZO*** as a descendant it too generates an ***IZO*** symbol. The coefficient 23 is neither significant nor does it have any significant descendants, which results in the generation of a single ***ZTR*** symbol for the entire tree. If any parent coefficient is allocated a symbol other than a ***ZTR*** then all of its immediate descendants will be allocated symbols. All other

coefficients are not assigned symbols and are not transmitted. Figure 3-12 and Figure 3-13 show the resultant trees for both the coefficients and symbols respectively. Where Z = ***ZTR***, I = ***IZO***, P=***POS*** and N=***NEG***.

Table 3-1: EZW Example First Pass

Subband	Coefficient Value	Symbol
LL	63	POS
LL	-34	NEG
LL	-31	IZ
LL	23	ZTR
HL	49	POS
HL	10	ZTR
HL	14	ZTR
HL	-13	ZTR
LH	15	ZTR
LH	14	IZO
LH	-9	ZTR
LH	-7	ZTR
LH2	7	ZTR
LH2	13	ZTR
LH2	3	ZTR
LH2	4	ZTR
HL2	-1	ZTR
HL2	47	POS
HL2	-3	ZTR
HL2	-2	ZTR

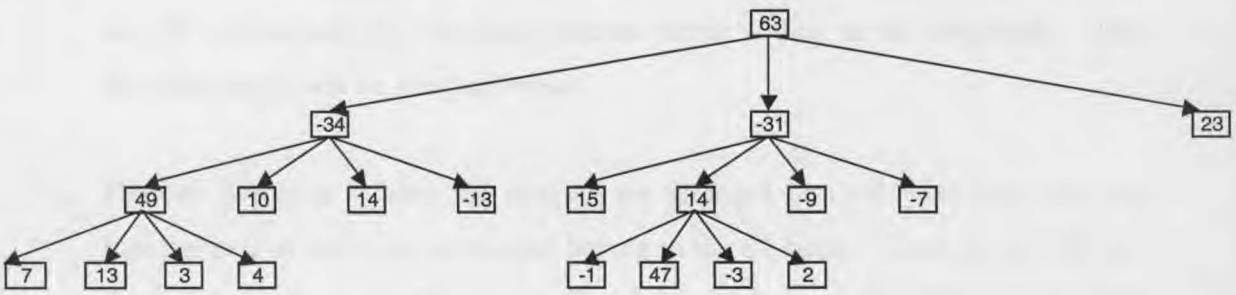


Figure 3-12: Example EZW Coefficient Tree

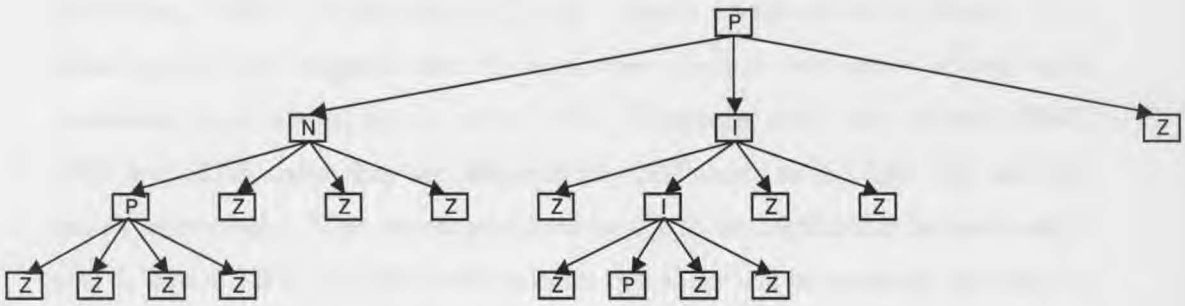


Figure 3-13: Example EZW Symbol Tree

These symbols are then scanned according to section 3.2.2, where the coefficients are first ordered in terms of subbands and then "zigzag" grouping of children. The transmitted contents follow the order in Table 3-1.

5. **Transmit Refinement Bits (Subordinate Pass)** – Since the four coefficients 63, -34, 49 and 47 are identified as significant, a further refinement bit can be sent in this pass. This is performed by evaluating (Eq. 3.2), where T is the current threshold and C_{sig} represents any previously identified significant coefficient magnitudes.

$$\left(C_{sig} - T\right) \geq \frac{T}{2}$$

(Eq. 3.2)

If this condition is met then a binary (1) is produced for that symbol, otherwise a binary (0) is produced. In this case the output symbols are 1, 0, 1 and 0 as $31 > 16$, $2 < 16$, $17 > 16$ and $15 < 16$ respectively. These bits are then reordered according to decoding priority, where coefficients that are decoded to higher values receive refinement bits from the start of the stream. Implementation of this requires that a decoder accompany the encoding process, hence adding to its complexity. The decoding stages will be presented next.

6. **Decode Symbols** – Since the symbols are arranged in a subband hierarchy, the first symbols to arrive or be decoded belong to the LL band. These symbols define the initial significant coefficients in the LL band. In turn they offer the decoder information about which coefficients are significant in the next most significant band, so as to allocate the next incoming symbols appropriately. In this example the first symbol is a ***POS***, which indicates that the LL coefficient is a positive coefficient, which is in the interval {32,64). Hence a value of 48 is chosen. This initial symbol also suggests that the next three symbols that arrive belong to its immediate descendants, as it is not a ZTR. Therefore, when the symbols ***NEG***, ***IZO*** and ***ZTR*** arrive they are allocated to coefficients in the LH, HL and HH bands respectively. The reconstructed values for these coefficients become –48, 0 and 0. The ***ZTR*** in the HH band indicates that there will be no more symbols for the HH range of subbands in this pass. The ***NEG*** and ***IZO*** symbols suggest that

the next eight symbols to arrive belong to the LH1, and HL1 subbands respectively. Figure 3-14 illustrates this process, where the light grey boxes indicate the positions of the next coefficients to arrive. The dark grey boxes drawn in the last tree represent coefficients that are anticipating refinement bits as described in the next section. Figure 3-15 firstly reiterates the band structure for reference and the final decoded coefficients after the first dominant pass.

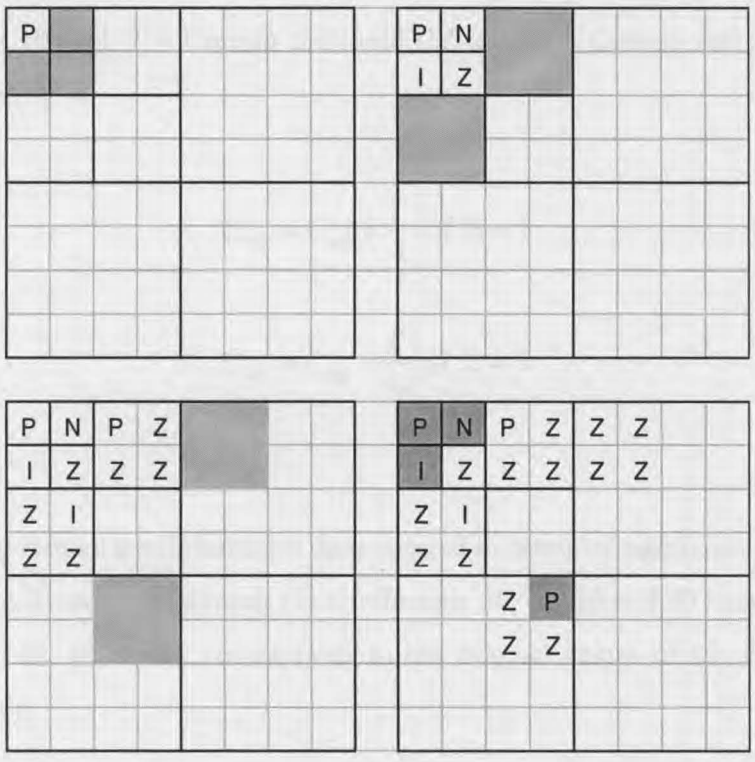


Figure 3-14: Example EZW Symbol Decode

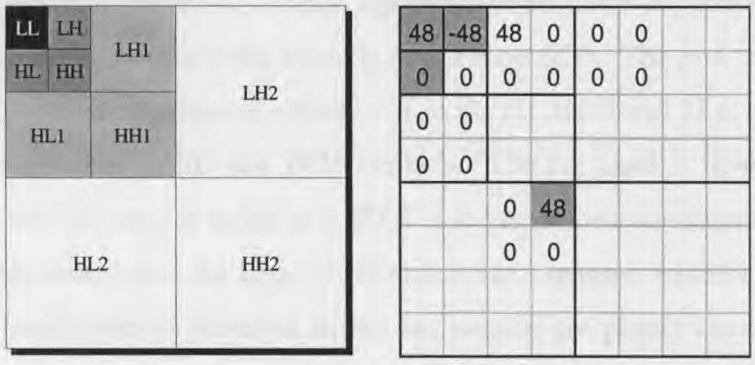


Figure 3-15: Example EZW after 1 Pass

7. **Refinement Decode** – Once the symbol pass identifies the coefficients to be refined then this process involves is the application of the incoming ordered stream of refinement bits to those coefficients. Since all the decoded coefficients so far are in the same range, the decoder expects the stream to be organised in terms of the subbands. Therefore it is ordered as LL, LH, LH1 and HL2. The incoming bit-stream of 1,0,1 and 0 refines the coefficients to 56, -40, 56 and 40 respectively. These values are calculated by evaluating (Eq. 3.3), where C_{sig} = Current significant symbol, T = Current threshold (32) and R = Current refinement bit for coefficient.

$$C_{sig} = C_{sig} + \frac{T}{4} \Big\} \text{if } R = 1$$

$$C_{sig} = C_{sig} - \frac{T}{4} \Big\} \text{if } R = 0$$

(Eq. 3.3)

The newly refined coefficients are then ordered in terms of significance for the next refinement decode. As a result the coefficients 56, 40, 56 and 40 becomes ordered as 56, 56, 40, 40, which corresponds to the original values of 63, 49, 34 and 47 respectively.

8. **Recalculate Threshold & Reiterate** – The threshold is now halved to result in the value 16 and the algorithm is repeated from step 3 but with the new threshold. This time however previously defined significant coefficients are treated as zero (0), which can only be assigned the symbols of ***ZTR*** or ***IZO***. The new threshold then identifies two new coefficients, namely –31 in the HL band and 23 in the HH band, which are coded as ***NEG*** and ***POS*** symbols. The LL band is now coded as an ***IZO***, but the LH band is coded as a ***ZTR*** as it has no new significant coefficients. For the refinement pass the new data is ordered in a manner where data for all the previous coefficients as arranged in the last section are placed ahead of the new coefficients. And the entire decode process is repeated.

3.3. ZTE Algorithm

In 1996 Stephen A. Martucci, Iraj Sodagar, Tihao Chiang and Ya-Qin Zhang [55] introduced a highly efficient zerotree wavelet based video coder for use in low bandwidth environments. The ZeroTree Entropy coder (ZTE) was originally submitted as a possible alternative for the primary video communications mechanism within the MPEG-4 standard. However, due to an abundance of available DCT based processors it was not considered for this purpose and was later chosen for the compression of textures within still images in the MPEG-4 standard [56].

The ZTE algorithm, like its predecessor (EZW), is based on the mapping of coefficients resulting from a DWT. It also exploits the self-similarity, or relations between the high and lower frequency subbands, to efficiently code large 2D areas of insignificant coefficients with symbols. The primary difference between the two relates to the way quantisation is performed. While the EZW performs a SAQ on the wavelet coefficients, the ZTE applies a single external quantisation process prior to coding the symbols, thus not supporting true embedded quantisation or multi-resolution coding. However, a significant advantage of this scheme is that it simply requires two passes, one for the symbols and the other for the coefficients. In addition a simple pseudo multi-resolution technique can be implemented by reordering the symbols and coefficients in an appropriate subband hierarchy. Martucci [55] also proved that this codec performed better for low bit-rate video communication with an added advantage that resulted in a more constant quality-rate.

The functional block diagram for a typical ZTE algorithm can be seen in Figure 3-16.

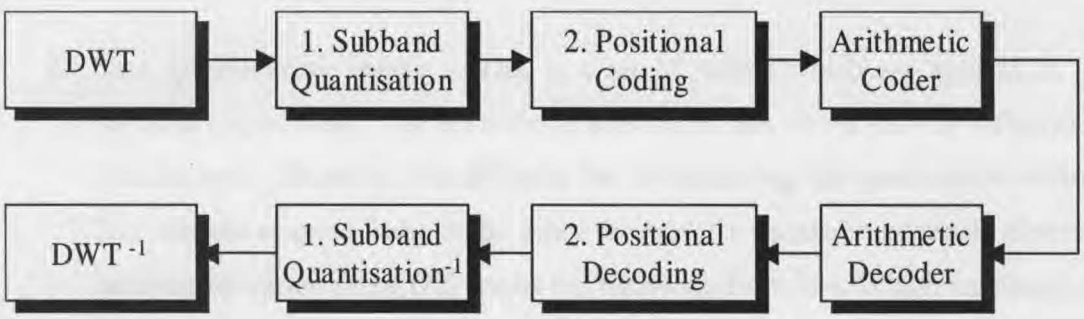


Figure 3-16: Main ZTE Components (Encoder & Decoder)

A description of aspects relating to the ZTE coding/decoding process is presented next.

3.3.1. Subband Quantisation

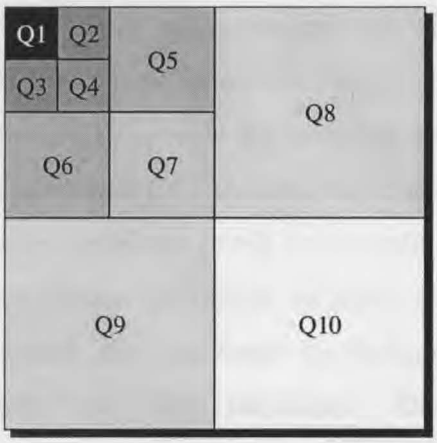
Since the ZTE algorithm does not include a fully embedded quantisation scheme, it depends on an external process to provide zero coefficients so as to build zerotrees. Typically a predefined quantisation matrix, either in the form of singular values for subbands or a fully qualified subband-sized quantisation array, is applied to each subband. The lossy-compression component for the ZTE algorithm occurs here and as such it directly controls the number of significant coefficients. The aim being to quantise as many high frequency coefficients to a zero value, which is typically performed by quantising these high frequency coefficients with much larger values than that used for low frequency coefficients. If an image has little or no high frequency components then the three largest bands, taking up ¾ of the image resolution can be quantised to near zero. The ZTE algorithm relies on the existence of a significant number of grouped zero coefficients to operate efficiently.

Without an embedded quantisation scheme this technique lacks the simple rate control mechanism (i.e. the bit-rate control) exhibited by the EZW algorithm. The rate control scheme here is dependant on two factors,

1. **The characteristics of the image** – Generally the lower the higher frequency content an image possesses the lower the resulting bit-rate is. For example, in an extreme situation a single colour image, such as a black screen generated during a scene change, generates coefficients which only reside in the lowest frequency band. Since the image received is typically uncontrolled the main quantisation mechanism focuses on the next factor.

2. **The quantisation vector** – This is a set of values which are applied to each subband respectively. The coefficients selected in this vector directly influence the final bit-rate. Therefore, the difficulty lies in optimising this quantisation vector to best suit the channel bandwidth. Since the wavelet transform generally observes a decaying spectrum trend [22] across the subbands from low to high frequency, it is exploited to form a quantisation vector increasing in quantisation value. Figure 3-17 illustrates the quantisation vector and its application to relative subbands on the transformed image. The lighter colours represent larger quantisation coefficients in comparison to the darker colours. As can be seen, Q2 and Q3 are

of equal importance (similar colour) yet Q4 is less important than both of them, therefore Q2 and Q3 are quantised less compared to Q4.



Quantisation Vector
[Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10]

Figure 3-17: Quantisation Vector assignment for Subbands

The magnitudes chosen for Q1…Q10 depend on the DWT employed. Multi-tap filter based DWT algorithms (e.g. Vilasonor, Daubechies 9-3) generally produce coefficients that are larger in magnitude than most 2/3-tap filter based DWT algorithms (eg. Triangular, Harr). An adaptive approach to the selection of these Q values generally presents an optimal solution. The technique used in [55] operates on the principal that an average bit-budget derived from the previous image in a sequence, is used to kerb the bit-budget, and ultimately the quantisation vector, for the current image. Therefore, to achieve a targeted bit-rate the average bit-budget is appropriately altered. Once derived, this average is further subdivided to allocate all subbands an individual bit-budget. Based on the previous quantisation coefficient, Q and the current bit-budget, a new Q value is generated for each of the subbands. By applying linear regression and extracting the coefficients of a first order autoregressive model, the quantisation weighting Q1..Q10 can be calculated. Generally the initial frame is processed using a default setting and corrected over time.

3.3.2. Positional Information Coding

The tree generation technique closely resembles that used in the EZW in that it also exploits the self-similarity between the higher and lower frequency subbands.

Therefore the same parent-child relationships are still used. Three major differences, however, contrast the two;

1. **The ZTE only generates one set of symbols** – during image encoding only a single set of symbols are ever generated. These symbols rely on the quantisation mechanism to provide the necessary zero coefficients. Unlike the EZW, the ZTE does not search for a set of different threshold values, instead it defines coefficients as either significant (<>0) or not (=0). Then trees are formed to account for the non-significant (quantised to zero) coefficients. Since a SAQ process is not performed, the significant coefficients are encoded whole after the preceding symbols have been identified. Distinctions between positive and negative significant coefficients are not made either. Therefore, an extra bit is embedded into the coefficient value to indicate the sign.

2. **Two pass mechanism per image** – The lack of SAQ mechanism enables the encoding and decoding process to be completed in just two passes. One for the symbols and the other for coefficients.

3. **Only 3 different symbols are used** – The ZTE scheme uses 3 different symbol types to identify the contents of a DWT. This smaller sized symbol set has been proven to perform better for low bit-rate video.

To account for all possible cases the following three symbols are used.

1. ***ZTR*** **– Zero Tree Root** – These symbols define coefficients are not significant, have significant parents, yet contain no significant descendants. The more ***ZTR*** symbols generated the better the compression.

2. ***VZT*** **– Valued Zero Tree root** – These symbols identify coefficients that are significant yet contain no significant descendants. This type of symbol is used to facilitate low bit-rate video. All significant coefficients in the highest frequency subband are allocated this symbol.

3. ***VAL* – A Value** – *Coefficients identified as **VAL** are determined when they contain significant descendants. As such an insignificant coefficient can be deemed a **VAL** if it contains a significant descendant.* In this case, where the decaying spectrum phenomenon is violated, a zero coefficient value is transmitted for this coefficient.

All other coefficients are zero and are not transmitted.

3.3.3. Encoding Process

62	-32	48	0	0	0	0	0
-28	20	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	32	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 3-18: [2 4 16 16] Quantised Coefficients

To illustrate the operation of the encoding process the example wavelet coefficients in Figure 3-11 are reused. After applying a subband quantisation vector of [2 4 16 16], for each of the Q values [Q1…Q4], on the wavelet coefficients, the resultant pre-encode map is presented in Figure 3-18. The following steps are then performed.

1. **Identify Significant Coefficients** – Each quantised coefficient, in the coefficient map is iteratively searched for significance determination. A coefficient is deemed significant if its magnitude is greater than zero. The significant coefficients for this example are shown Figure 3-19.

2. **Identify Parent / Child Relationships** – The significance map is then arranged in a tree structure identifying all parent and child relationships inherent to

hierarchical wavelet decompositions. The tree structure, for this example, also containing the significant coefficients, is presented in Figure 3-20.

S	S	S	0	0	0	0	0
S	S	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	S	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 3-19: Example ZTE Significance Map

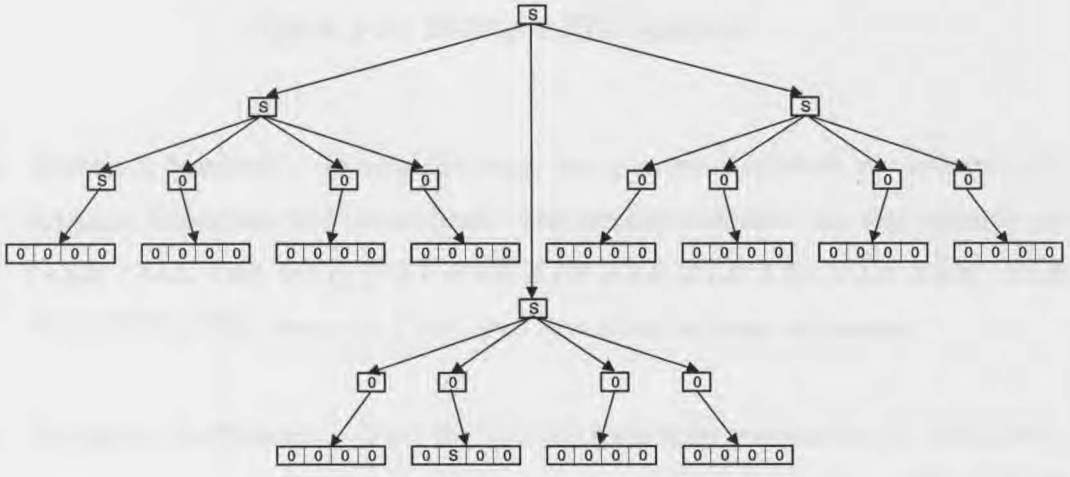


Figure 3-20: Example ZTE Significance Tree

3. **Generate Symbols** – Each relevant coefficient in the map is assigned a symbol, otherwise it is not transmitted. Significant coefficients that have no significant descendants are assigned the ***VZT*** or valued zero tree root symbol. All other significant coefficients are assigned as ***VAL***. Insignificant coefficients with significant descendants are assigned the symbol ***VAL*** with coefficient magnitude 0. All insignificant coefficients that belong to a significant parent yet contain no significant descendants are assigned the ***ZTR*** symbol. Figure 3-21 illustrates the resultant symbol tree for this example (the following shortened forms are used V =

VAL, VZ = ***VZT*** and Z = ***ZTR***). The areas without any symbol allocation are not transmitted.

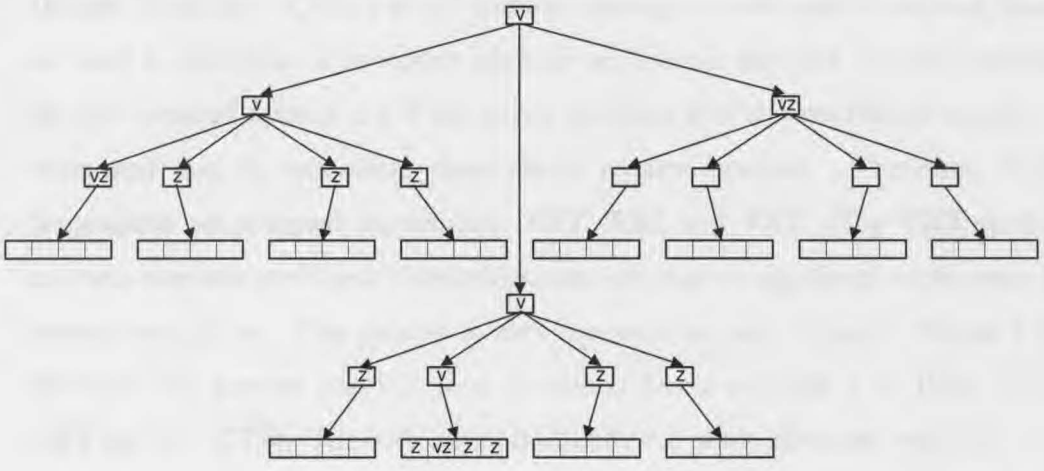


Figure 3-21: Example ZTE Symbols

4. **Transmit Symbols** – During this stage the generated symbols are ordered into subband hierarchies and transmitted. The arranged symbols for this example are [***VAL***] [***VAL VAL VZT***] [***VZT ZTR ZTR ZTR ZTR VAL ZTR ZTR***] [***ZTR VZT ZTR ZTR***], where the [ ] are used to indicate subband separation.

5. **Transmit Coefficients** – Once the symbols have been transmitted the coefficients are ordered into subbands and transmitted. For this example they are [62] [-32 –28 20] [48 0] and [32]

This step concludes the two-pass ZTE coding process for a particular image.

3.3.4. Decoding Process

The decoding process, like that performed in the EZW, commences once symbols for the lowest subband have been received. Since these symbols are hierarchically arranged, symbols from a lower frequency subband are used to allocate incoming symbols to the next higher frequency subband. A step-wise description of the decoding process with input vectors [***VAL***] [***VAL VAL VZT***] [***VZT ZTR ZTR ZTR***

ZTR VAL ZTR ZTR [***ZTR VZT ZTR ZTR***] for the symbols, and [62] [-32 –28 20] [48 0] [32] for the coefficients follows.

1. **Decode Symbols** – Once a set of symbols relating to a subband is received, these are used to determine if any other symbols need to be decoded. In this example the first received symbol is a ***VAL***, which indicates that this coefficient requires a value and that its immediate descendants require symbols. Therefore, these descendants are assigned the symbols ***VZT***, ***VAL*** and ***VZT***. The ***VZT*** symbol indicates that this coefficient's descendant tree contain no significant coefficients to expect symbols for. This process is then repeated for each subband. Figure 3-22 illustrates this process (the following shortened forms are used V = ***VAL***, VZ = ***VZT*** and Z = ***ZTR***). The bold symbols identify the newly allocated symbols.

V	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?

V	**V**	?	?	?	?	?	?
V	**VZ**	?	?	?	?	?	?
?	?	0	0	?	?	?	?
?	?	0	0	?	?	?	?
?	?	?	?	0	0	0	0
?	?	?	?	0	0	0	0
?	?	?	?	0	0	0	0
?	?	?	?	0	0	0	0

V	V	**VZ**	**Z**	0	0	0	0
V	VZ	**Z**	**Z**	0	0	0	0
Z	**V**	0	0	0	0	0	0
Z	**Z**	0	0	0	0	0	0
0	0	?	?	0	0	0	0
0	0	?	?	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

V	V	VZ	Z	0	0	0	0
V	VZ	Z	Z	0	0	0	0
Z	V	0	0	0	0	0	0
Z	Z	0	0	0	0	0	0
0	0	**Z**	**VZ**	0	0	0	0
0	0	**Z**	**Z**	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 3-22: Example ZTE Symbol Decode

2. **Decode Values** – In this phase coefficients that were assigned significant symbols in the previous stage are allocated coefficient values from the stream. As per the symbols these are arranged in a hierarchical subband stream. With this example the first decoded coefficient, the LL coefficient, receives the value 62. The next set of subband coefficients receives values –32, -28 and 30. In this manner the whole image is reconstructed. Figure 3-23 depicts the coefficient value decode process.

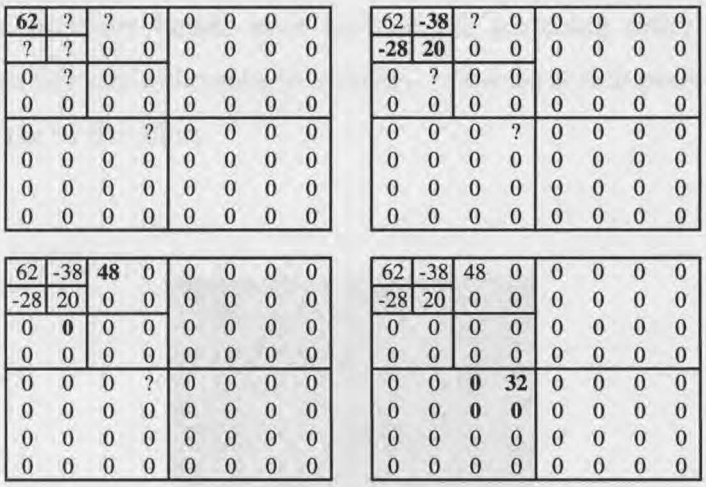


Figure 3-23: Example ZTE Coefficient Value Decode

Completion of this process cycle results in singe image decode.

3.4. Search Techniques

The parent-child significance determination process, especially when considering implementation, deserves particular attention as it dictates the memory structure and processing pattern required to perform the identification. Two typical techniques for such a search are described below.

3.4.1. Depth First Search

The Depth First Search (DFS) [57] is a significance identification search that scans a dependence tree in a bottom-up manner, selecting a single sub-tree at a time. It requires that the coefficient data be organised into sub-trees, with the highest frequency subbands located at the outer fringes of the main tree, and the lowest frequency subbands located towards the trunk of the main tree. Once a sub-tree has been scanned the next sub-tree is selected for scanning, until all the sub-trees are scanned, whereupon the main trunk linking all the sub-trees is scanned. Figure 3-24 is a typical wavelet coefficient map that allocates a unique label for each coefficient. The significance dependence tree for this map is shown in Figure 3-25. The dark curved

arrows and the numbers beside each indicate the searching order. For a single processor system this method results in a vertical tree-wise search performed from one horizontal extreme to the other.

aa	ab	ba	bb	ea	eb	fa	fb
ac	ad	bc	bd	ec	ed	fc	fd
ca	cb	da	db	ga	gb	ha	hb
cc	cd	dc	dd	gc	gd	hc	hd
ia	ib	ja	jb	ma	mb	na	nb
ic	id	jc	jd	mc	md	nc	nd
ka	kb	la	lb	oa	ob	pa	pb
kc	kd	lc	ld	oc	od	pc	pd

Figure 3-24: Coefficient Map

A typical realisation of the DFS technique requires the use of a single linea bank of RAM, arranged into trees as in Figure 3-26, and a few variables to retain significance information. A pointer keeps track of the current coefficient being processed, which are organised into groups of four. Each group supplies significance information to its parent. When a parent is analysed, its significance information is stored for the next level parent in an external variable for that scale. In this manner the entire image can be processed by a single sequential processor system. However, this zerotree search is more suited to a parallel processing system where each tree is processed individually as they have very little dependence on each other. When all sub-processors have completed the allocated task, a main processor can then collate the significance information.

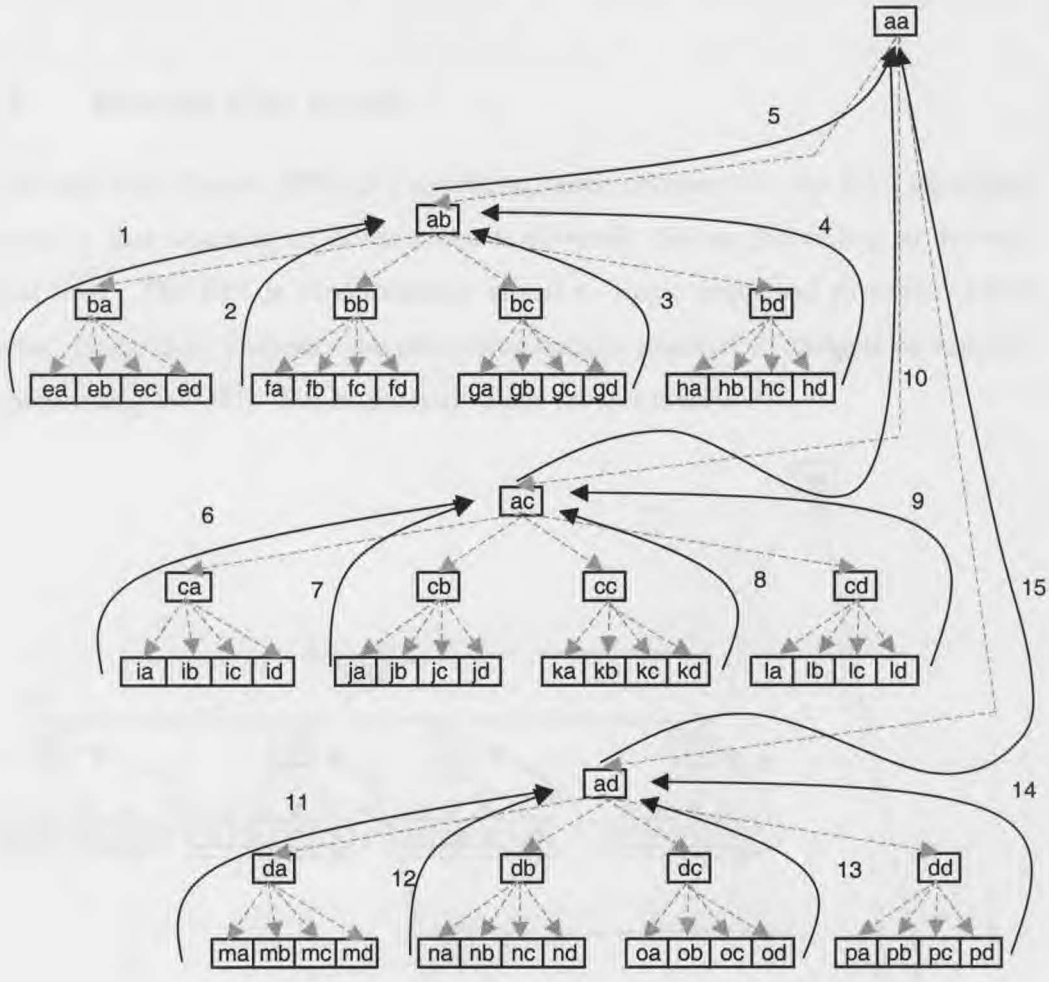


Figure 3-25: DFS Tree Search

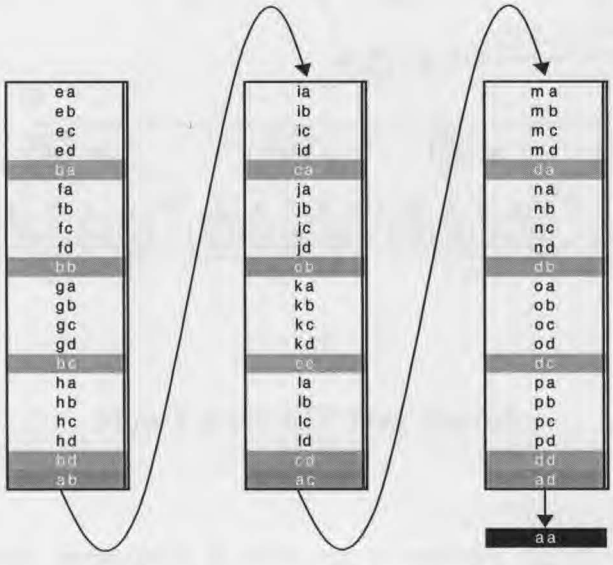


Figure 3-26: DFS Coefficient Arrangement

3.4.2. Breadth First Search

The Breadth First Search (BFS) [57] algorithm, when compared to the DFS algorithm, operates by first scanning an ordered tree horizontally before proceeding to the next vertical level. The BFS is predominantly suited to single sequential processor based systems. Figure 3-27 illustrates the processing pattern required to navigate an example tree performing the BFS. The horizontal search pattern is seen.

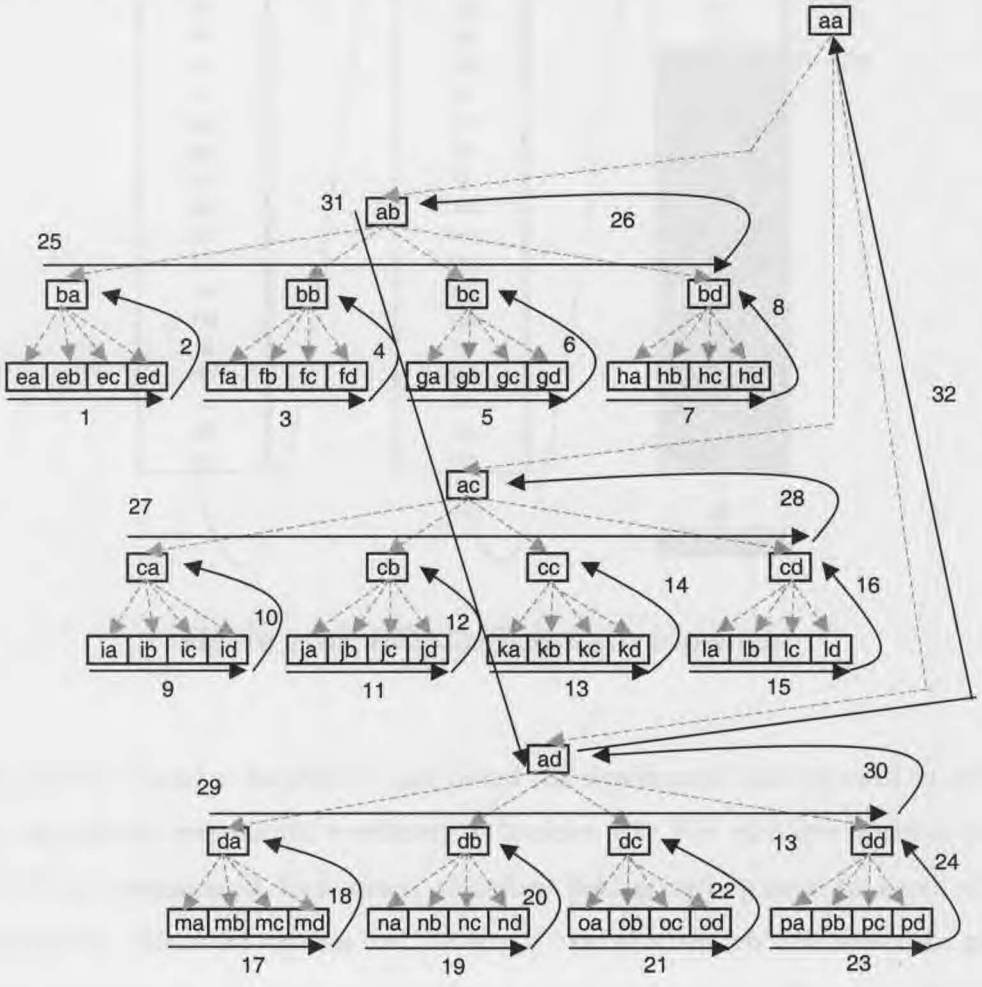


Figure 3-27: BSF Tree Search

The BFS technique, particularly in reference to zerotree significance identification, is typically realised by the use of a single bank of RAM, which is coupled together with two index pointers. The coefficients are firstly arranged into the manner depicted in

Figure 3-28. One of the two pointers (P1) is initialised at coefficient 'ea' or the start of the block of memory, while the other (P2) is initialised to 'ba' or the start of the next level.

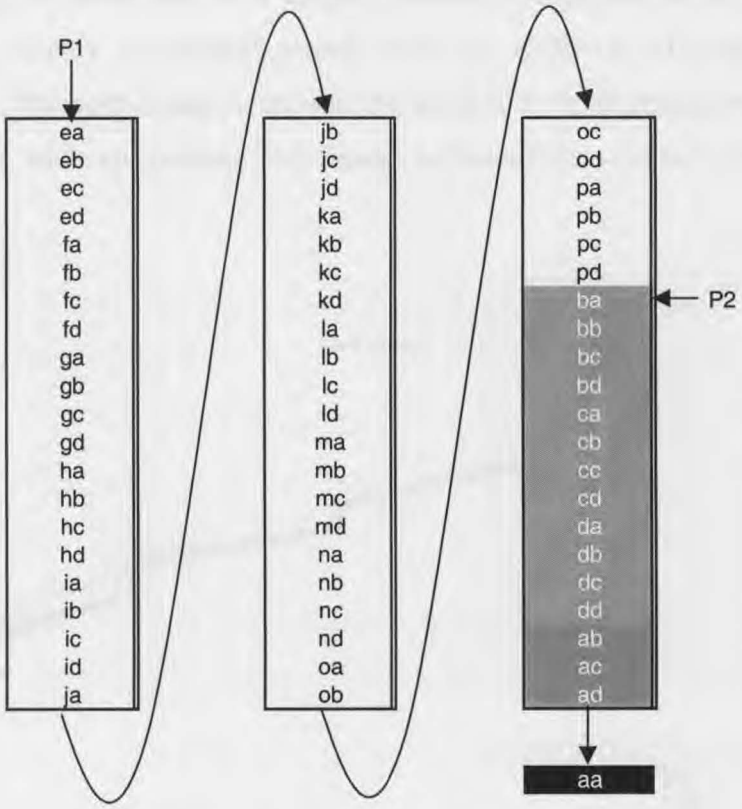


Figure 3-28: BFS Coefficient Arrangement

Coefficients found at location P1 are tested for significance and are used to establish the dependence tree for the coefficient at location P2. For each increment in pointer P2, P1 is incremented four times, therefore linking each parent to a set of four dependants. When P1 reaches 'ba', P2 would have reached 'ab' and hence the parent-child relationship is correctly matched for the next higher level. The only exception to this is the lowest scale 'aa', which is related to only three dependants and as such P1 is only incremented three times when P2 points to 'aa'.

The main advantage of the BFS technique is that it facilitates the use of a single processor to perform an efficient encode.

3.5. Performance Results

To test the suitability of both algorithms, two images were selected as inputs. The 512x512 pixel 'Lena' image to represent common images and the 176x144 pixel 'Jenny' image more suited to a typical representation of a mobile multimedia communications image. Both coding methods employ a triangular wavelet filter, an EZW or ZTE scheme and an arithmetic coder. The performance comparison is attained by compressing each image to a variety of bit-rates, while ascertaining Peak Signals to Noise Ratios for each such rate.

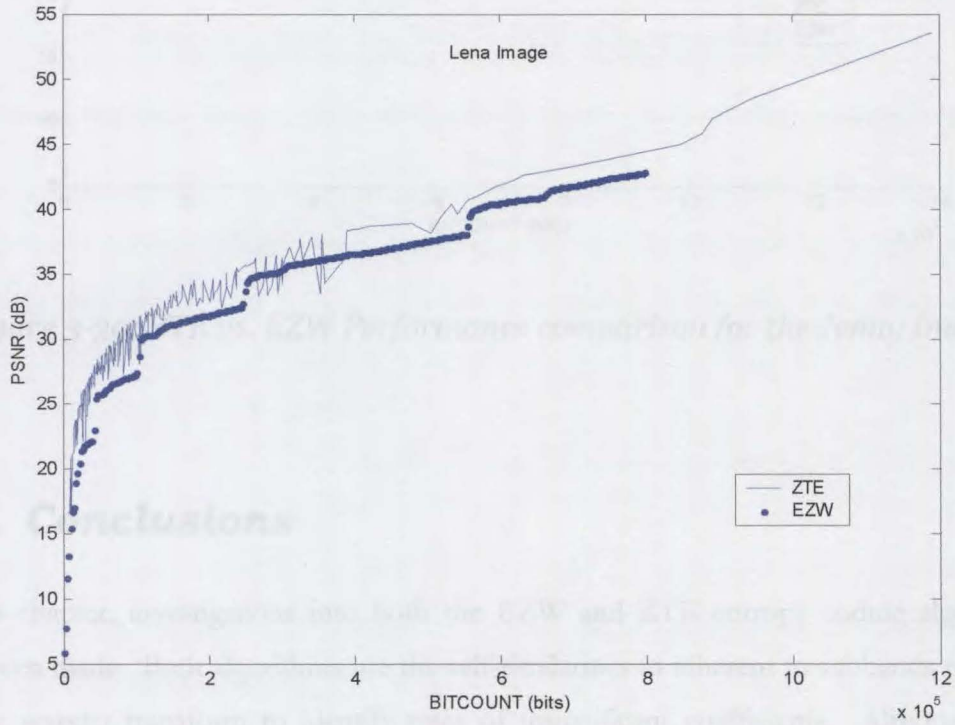


Figure 3-29: ZTE vs. EZW Performance comparison for the Lena image

Figure 3-29 displays the performance curves for both ZTE and EZW algorithms when applied on the Lena image. A marginal improvement in the PSNR for the ZTE algorithm especially at lower bit-rates can be observed. In Figure 3-30, the performance graph for the 'Jenny' image, this improvement is also visible.

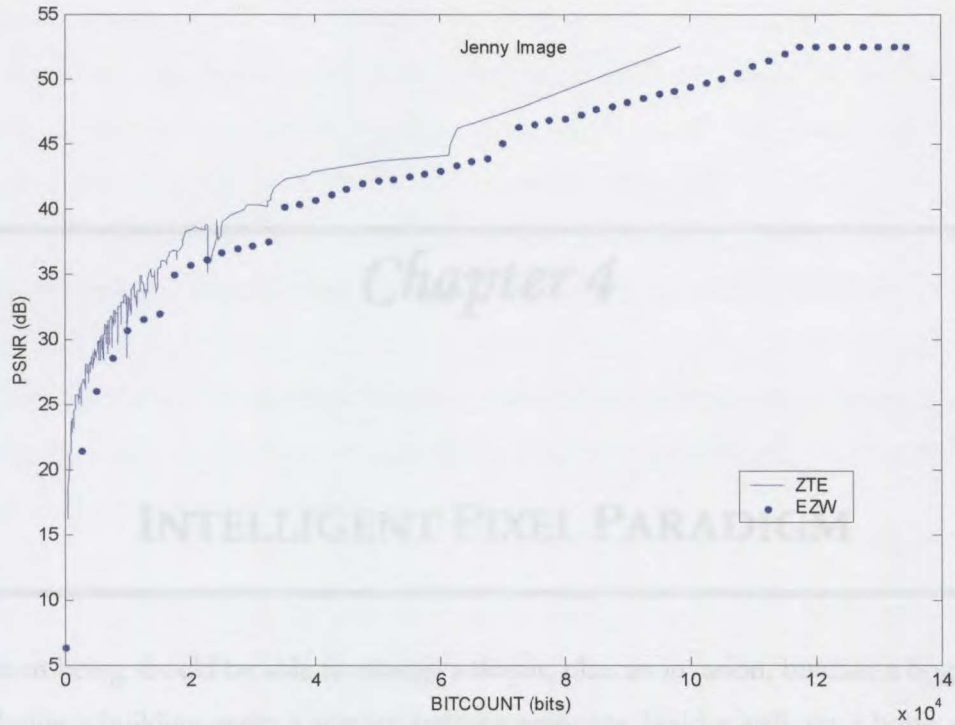


Figure 3-30: ZTE vs. EZW Performance comparison for the Jenny Image

3.6. Conclusions

In this chapter, investigations into both the EZW and ZTE entropy coding algorithms have been made. Both algorithms use the self-similarities in inherent to subbands resulting from a wavelet transform to identify trees of insignificant coefficients. Although both techniques are similar in nature, the differences that separate them make for key consideration especially for hardware implementation purposes. The EZW algorithm has advantages in terms of precise rate control through the use of multiple symbol and refinement data passes. In comparison, the dual-pass nature of the ZTE, however, improves the coding speed and simplicity of implementation, particularly in reference to hardware implementation. The marginal improvement in performance exhibited by the ZTE algorithm should also be noted. As neither algorithm has a clear advantage over the other, the task of selecting the most appropriate for parallel hardware implementation is reserved for Chapter 5, where hardware issues of both algorithms are examined.

Chapter 4

INTELLIGENT PIXEL PARADIGM

"A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyse a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. Specialization is for insects."

Lazarus Long, "Time Enough For Love" (R.A. Heinlein)

4.1. Introduction

The Intelligent Pixel Paradigm (IP), first introduced by Professor Kamran Eshraghian in 1998 [58], is a novel OPTO-VLSI architecture targeted towards low-power multimedia communications systems. In comparison to conventional multimedia communications equipment, this paradigm attempts to amalgamate traditionally modular multimedia system components, i.e. capture, processing & display, into a single device. Although applicable to a variety of systems, this paradigm was initially considered ideal for video compression/decompression systems. In this case, the IP paradigm introduces a further novel component that suggests that the multimedia system be realised by a parallel, independent, pixel-wise processor array termed the Intelligent Pixel Array (IPA). The potential advantage of this technique is two fold. Firstly it allows for array size scalability to any arbitrary size (i.e. the array size is independent of the processing architecture) with

minimal increase in clock frequencies, secondly employing a massively parallel processing architecture allowing for the use of very low clock speeds, providing the same processing ability at a lower cost in required power. The major challenge posed by this paradigm is two fold. Firstly the challenges posed in designing a traditionally sequential processor based algorithm in a massively parallel environment. Secondly the limitations posed by VLSI design technologies such as the maximum die-size, minimum feature size and production yields (i.e. larger designs increase the possibility producing failures).

This thesis focuses on the development of a massively parallel pixel-wise zerotree coding / decoding algorithm to suit the novel Intelligent Pixel Array architecture for the purpose of multimedia video communications.

4.2. Conventional Video Coding Systems

Traditionally, multimedia video communications systems are composed of five primary components.

1. **CCD (Charge Coupled Device), CMOS or other video camera** – for image capture at a specified rate (video).
2. **ADC(Analogue to Digital Converter) and Encoding** – Converts an analogue video signal to digital before applying Digital Signal Processing (DSP) techniques to compress the size requirements of the captured video.
3. **Transmitter/Receiver** – This component handles the communications between two multimedia video communications devices within a specified bandwidth limitation and standard (i.e. 3G, GSM etc.).
4. **Decoder and DAC (Digital to Analogue Converter)** – Expands a compressed video stream to original bandwidth, before converting this to an appropriate analogue signal for display.
5. **CRT, LCD, LED or other display device** – for the display of decoded images.

Conventional multimedia solutions [72] generally isolate these components into separate devices (eg Microsoft's ASF streaming video format).

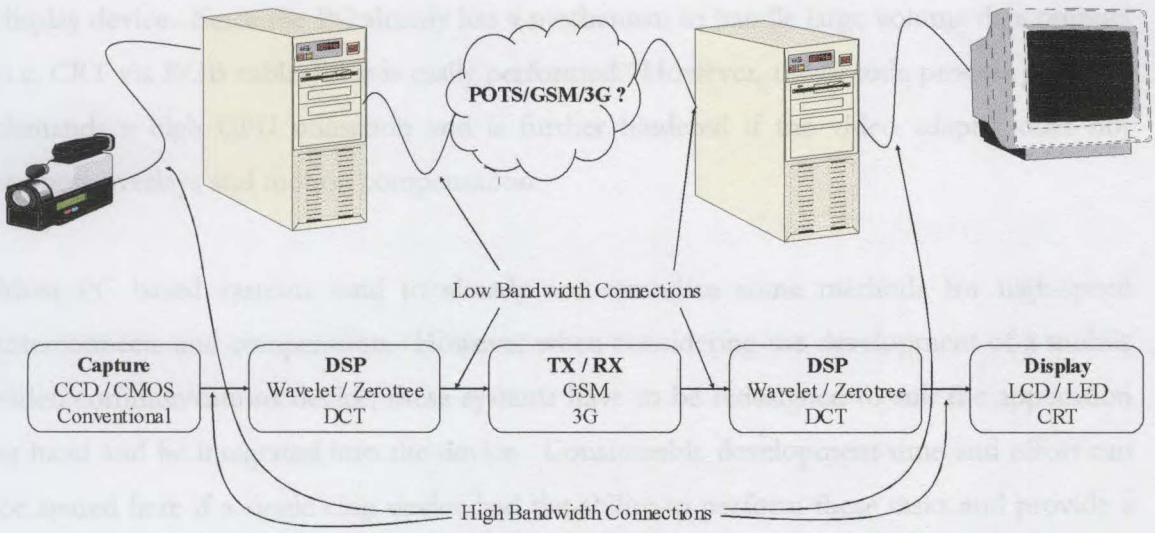


Figure 4-1: Conventional Video Communication Methodology

Figure 4-1 illustrates a typical multimedia communications system resembling that found in most PC based solutions. The main disadvantage of this system results from the existence of some high bandwidth interconnections between devices. This requires high speed interfacing between, for instance, the camera and DSP device. For example a QCIF image, with size 176 x 144, with an 8-bit colour depth, operating at 25 frames per second requires a bandwidth of approximately 5 Mbps. Although present technologies accommodate bit-rates of this order, as the resolution and colour-depth are increased, most systems, unless specially designed, perform inadequately. Generally in order to minimise the amount of data generated and exchanged between these components an analogue signalling technique is also employed (e.g. Composite, SVIDEO etc.).

The Digital Signal Processor (DSP) generally performs a transformational task (DCT, Wavelet etc.) on the raw data to produce a compressed, low-bandwidth stream which is then transmitted. However most PC based systems cannot adequately provide a high bandwidth interface while maintaining real-time compression of the raw data, especially at higher resolutions and colour depths. Here pure hardware solutions (eg. Matrox RT2500) that generates compressed streams tend to perform more efficiently.

Once compressed, the low bandwidth stream, targeted for a particular transmission channel is distributed. The decoding process receives a compressed data stream from the

network, which is then decoded by the DSP before producing the decoded frames on the display device. Since the PC already has a mechanism to handle large volume data outputs (i.e. CRT via RGB cables) this is easily performed. However, the decode process generally demands a high CPU utilisation and is further hindered if the video adapter does not support overlays and motion compensation.

Most PC based systems tend to already accommodate some methods for high-speed interconnects and compression. However when considering the development of a mobile video communications device, these systems have to be redesigned to suit the application at hand and be integrated into the device. Considerable development time and effort can be spared here if a single chip device had the ability to perform these tasks and provide a compressed stream for transmission. Devices that exhibit low power consumption characteristics will prove to be of particular use.

The scope for the IPA, therefore, can be summarised into the development of a massively parallel, low power, video capture, compression and display system implemented on a single chip.

4.3. The IPA Video Coding System

The Intelligent Pixel Array (IPA) is a concept that merges optical and VLSI technologies to provide a singular OPTO-VLSI device with the ability to provide the full functionality of conventional video communications systems. The "Divide and Conquer" principal has always been a particularly advantageous approach for the simplified realisation of any complex system, however in this case the merits of a more unified approach are exploited. Two major advantages of such a unified approach, especially in terms of realisation within a single VLSI device, are

1. **The development of a single system on chip solution** – for multimedia communications devices, simple single chip solutions provide an avenue for product size compaction and development cost reduction, for instance, for use in hand held units, possibly even wristwatches.

2. **Minimisation of interfacing between third party modules** – resources utilised in the development of a working interface between the typically high bandwidth capture and display components with processing components can be allocated for other needs.

To comply with this unified strategy, the IPA design attempts to incorporate the three components of image capture, processing and display into one 3D OPTO-VLSI device. The 3D component is derived due to the raw data I/O handling mechanism, which receives / transmits raw data in a perpendicular plane to the processing array as seen in Figure 4-2. The proposed video communications system supported by the IPA is illustrated in Figure 4-3.

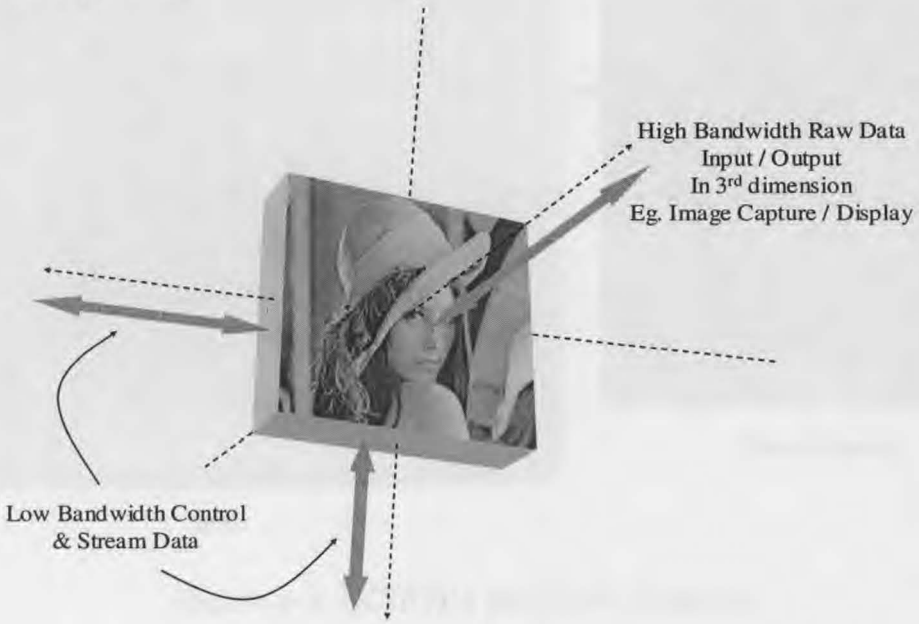


Figure 4-2: 3D Chip I/O

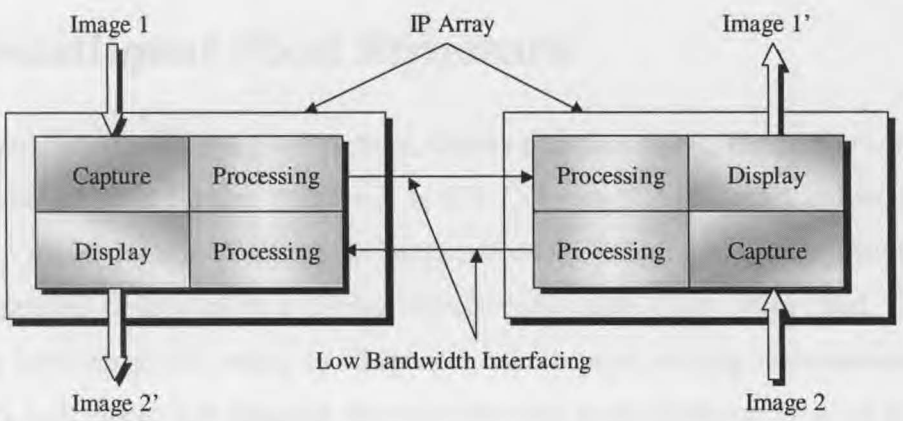


Figure 4-3: IPA Video Communications System

In order to accommodate a novel 3D I/O approach, the array design employs a pixel-wise capture mechanism, directly coupled to a massively parallel, pixel-wise processing array and a pixel-wise display element. The array configuration for a proposed QCIF sized device with a close-up of the visual pixel pattern is illustrated in Figure 4-4. The dark lines are caused by the inter pixel routing paths, while the dark squares at the top left corner of each pixel, is caused by the capture photodiode. The reflective metal surrounded by these dark artefacts constitutes the display component. The processing remains hidden under this display metal.

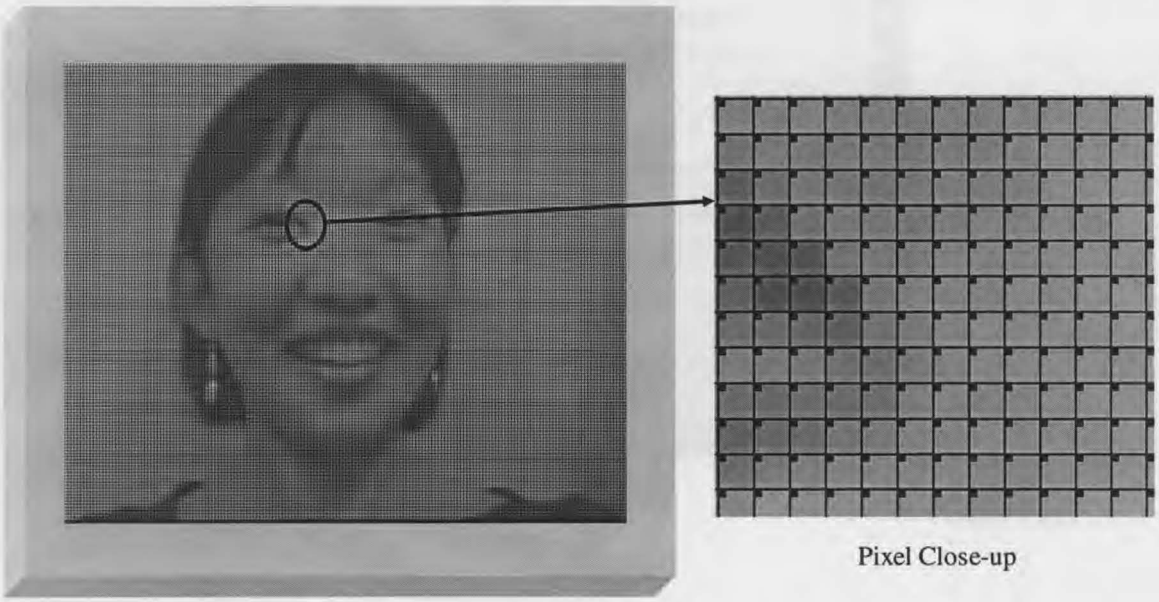


Figure 4-4: QCIF IPA and Pixel Close-up

4.4. Intelligent Pixel Structure

To accommodate parallel per-pixel capture, display and processing, the IPA is constructed with an array of M x N (for QCIF this is 176 Columns x 144 Rows) individual pixels arranged in a grid pattern. Each of the pixels, although differing in spatial position, share identical capture, processing and display characteristics with every other pixel. The only difference between pixels, relate to Chapter 5, where pixel routing interconnects differ. Figure 4-5 and Figure 4-6 illustrate the top-view and cross-sectional view of each pixel

respectively. Using this mentality, the image is captured via the photodiode, processed by the 'hidden' internal circuitry and displayed via the LCD component.

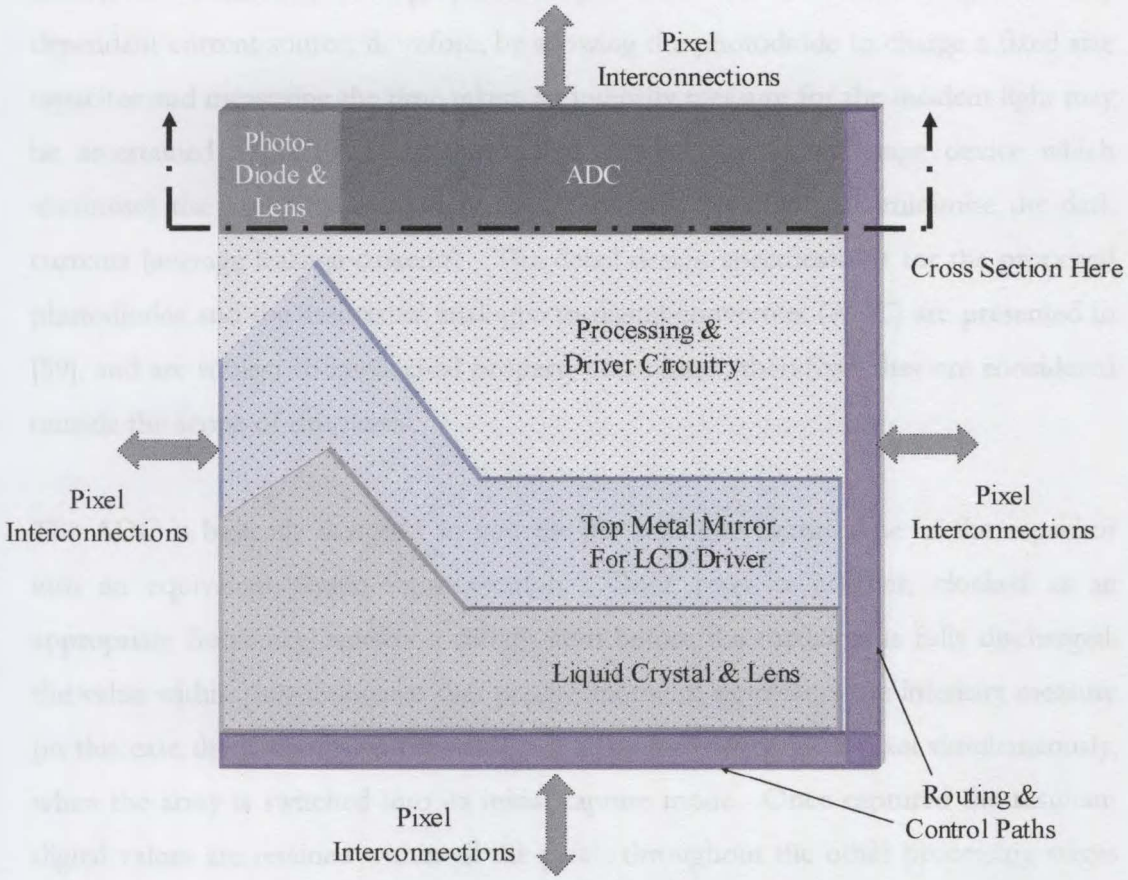


Figure 4-5: IP Top View

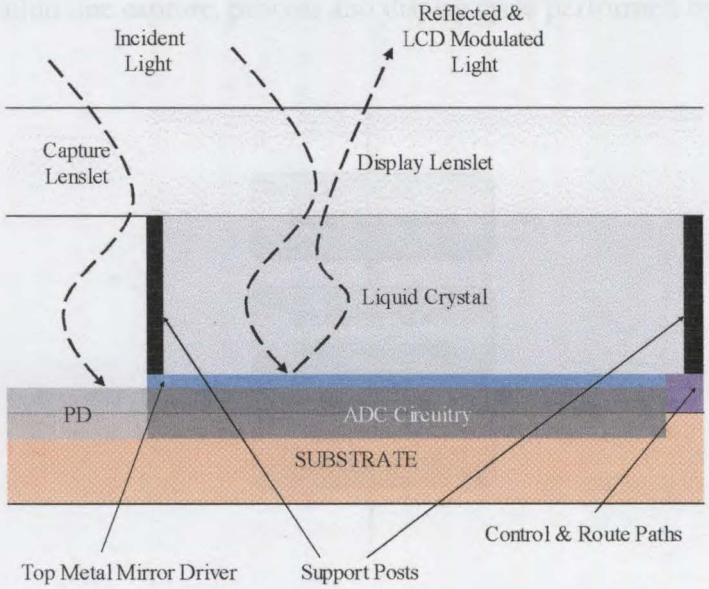


Figure 4-6: IP Cross-sectional View

4.4.1. IP Capture Component

In order to capture an image, the intensity of the light falling on the photodiode is measured. For all intents and purposes the photodiode can be viewed as light intensity dependant current source; therefore, by allowing the photodiode to charge a fixed size capacitor and measuring the time taken, an intensity measure for the incident light may be ascertained. The photodiode itself is designed as a two stage device which maximises the carrier generation (increased capture visibility) and minimise the dark currents (average leakage currents). The detail design specifications for the proposed photodiodes and the associated analogue to digital converters (ADC) are presented in [59], and are subject to intellectual property constraints; therefore, they are considered outside the scope of this thesis.

The ADC is basically designed to convert the analogue charge time of the capacitor into an equivalent digital representation. Once reset, a counter, clocked at an appropriate frequency, reaches a certain limit before the capacitor is fully discharged; the value within this counter at that precise moment, represents the intensity measure (in this case the counter is 6-bits wide). It is performed on each pixel simultaneously, when the array is switched into its initial capture mode. Once captured the resultant digital values are retained within all the pixels throughout the other processing stages until it is required to capture the next frame. This takes place exactly 40μs (for 25 fps) after the last capture. Figure 4-7 shows the basic steps performed by the capture component within one capture, process and display cycle performed by the array.

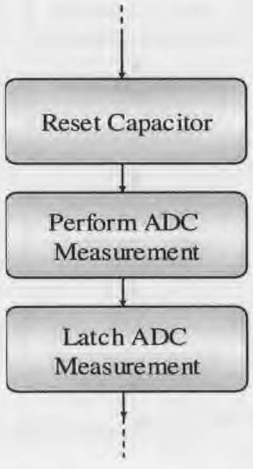


Figure 4-7: Capture Flowchart

4.4.2. IP Processing Component

Each pixel contains an identical processing component that is controlled via a single set of external control lines. A pixel, when in processing mode, operates through three basic modes, namely **Local Stream Encode, Local Stream Decode** and **Foreign Stream Decode**. In local stream encode mode the processor performs the primary task of encoding a raw image for transmission. In local stream decode mode the processor reconstructs the image as would be reconstructed at another such device. This allows for accurate motion analysis determination for the next frame. In the final stage, the foreign stream decode mode, the processor decodes the stream received from another such device in preparation for display.

These three basic modes further sub-divide into 15 sub-modes, which are based on the processing components associated with the video codec. The flowchart illustrating the order of these 15 sub-modes and their positions within the three basic modes are presented in Figure 4-8.

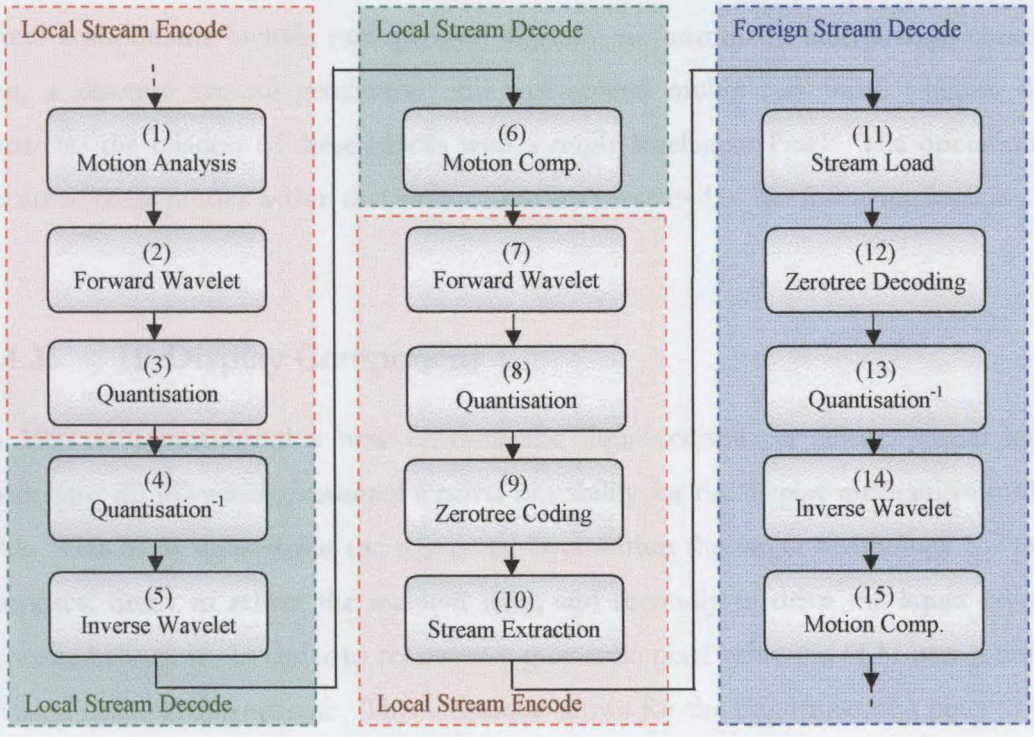


Figure 4-8: Pixel Processing Flowchart

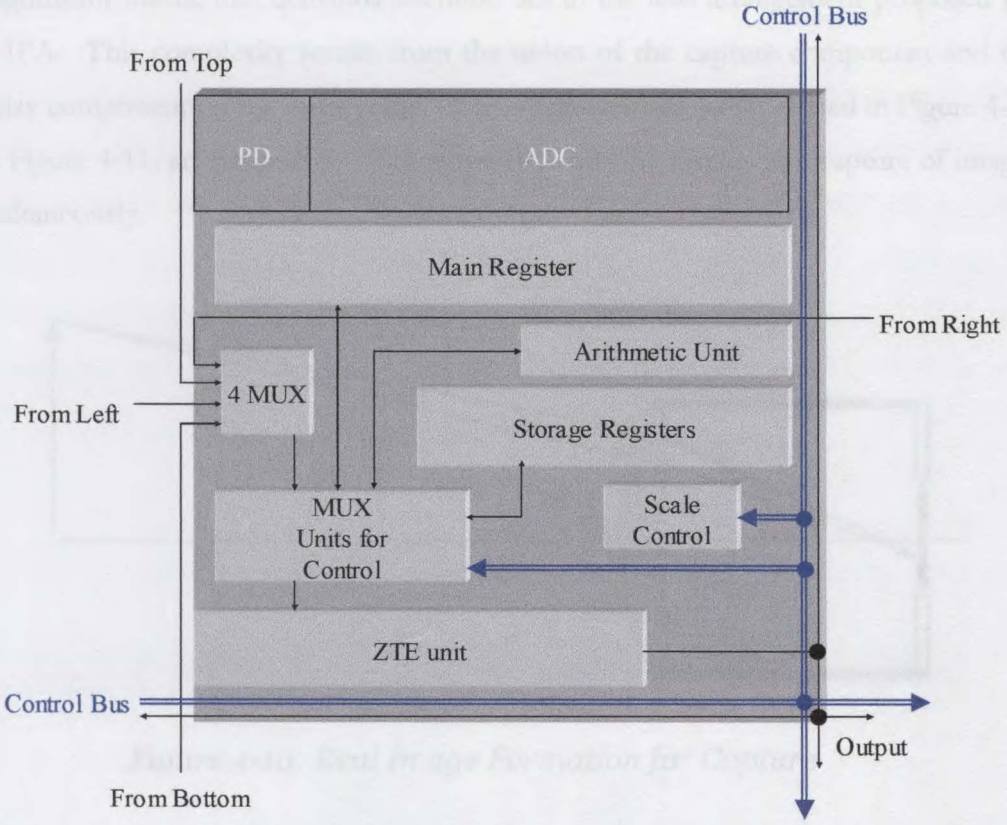


Figure 4-9: IP Processing Architecture

Several reusable components are employed to perform these 15 processing tasks. These components include multiplexers, registers, an arithmetic unit, a scale control unit, a zerotree symbol generation unit and several multiplexer units. Figure 4-9 illustrates the relation of these blocks with a single Intelligent Pixel. The operational details of these modes within this architecture are presented in the following sections.

4.4.3. IP Display Component

In 1997 [60] introduced a new ferro-electric liquid crystal on silicon spatial light modulator (SLM) which presented a novel possibility for the display mechanism in the IPA. This SLM utilises that the top metal layer within the target technology for two purposes, firstly to reflect the incident light, and secondly to drive the liquid crystal deposited above it. In order to represent a grey-scale pixel value the SLM uses a time-division multiplexing scheme. This technique allows for the brightness of a pixel to be related back to the coefficient value held within the pixel [61].

A significant hurdle that demands attention lies in the lens arrangement proposed for the IPA. This complexity results from the union of the capture component and the display component on the same plane. A lens arrangement, as illustrated in Figure 4-10 and Figure 4-11, are proposed. This allows for both the display and capture of images simultaneously.

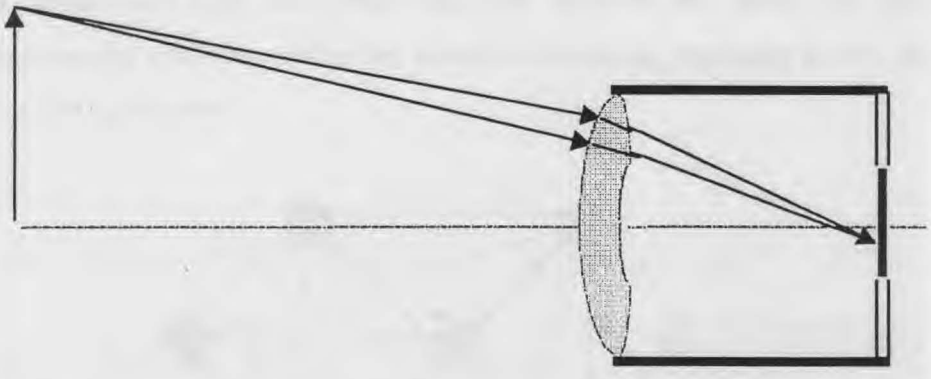


Figure 4-10: Real Image Formation for Capture

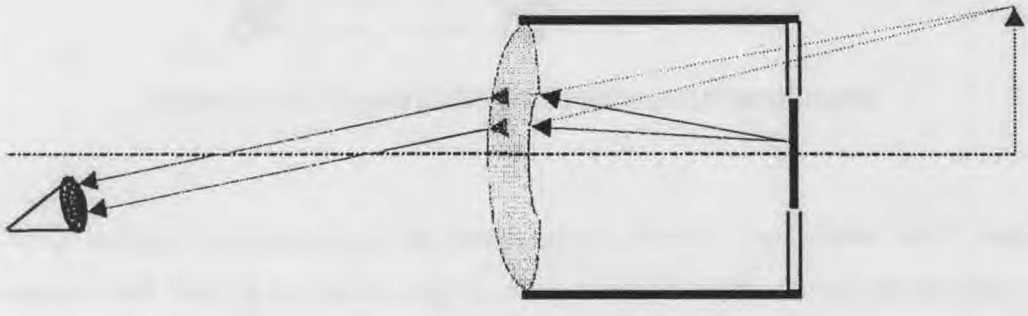


Figure 4-11: Virtual Image Formation for Display

Research in regards to this lens technology is currently being conducted at the University of Cambridge and due to intellectual property constraints full details relevant to this section is considered out of scope for this thesis.

4.5. IPA Interconnects

To be useful, all parallel processing environments require that its processing elements (PEs) such as the Intelligent Pixel (IP) be interconnected to other PEs for information exchange. For instance the hypercube architecture (as employed by C-Cube systems) suggests that each PE be connected to every other PE in the system [62], see Figure 4-12. Although this provides a flexible and high throughput data pipe between two pixels, the associated routing requirements tends to render this scheme impractical, especially in the case of a QCIF (176 x 144) pixel array.

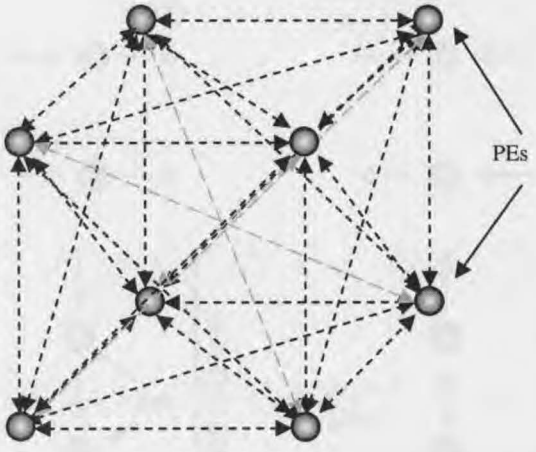


Figure 4-12: Hypercube Interconnect Arrangement

The IPA, instead, accommodates an interconnect scheme that allows each pixel to interconnect with four of its nearest neighbouring pixels in a lattice arrangement similar to that used in [63]. See Figure 4-13.

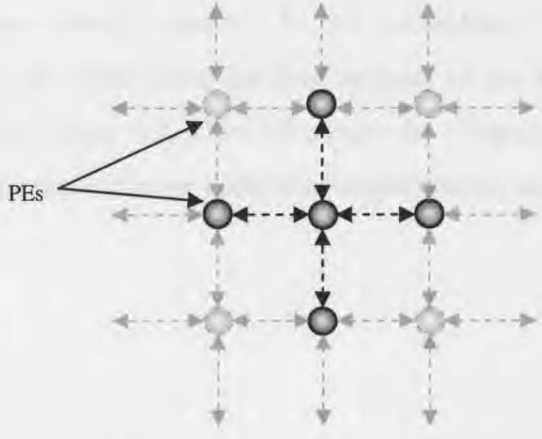


Figure 4-13: IPA Interconnect Arrangement

The communication direction of each pixel is then controlled via two global control signals that traverse to each pixel in the array. Therefore, at a particular instant data exchange is limited to only function in one direction, either left, right, up or down, throughout the entire array. The direction depends on the levels of the two signals *RC* (Row/Column selection) and *LR* (Left/Right selection). Figure 4-14 illustrates the four directions used and their corresponding signal levels.

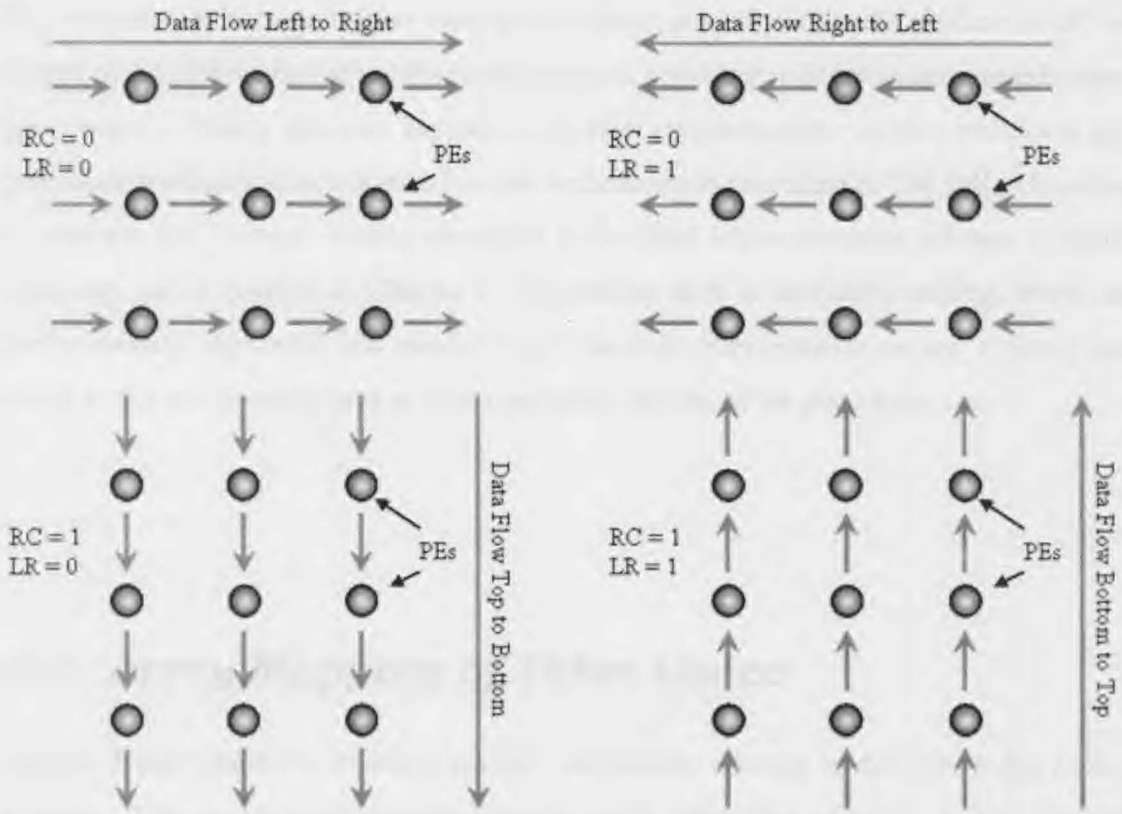


Figure 4-14: IPA Data Flow Directions

The direction control lines directly control a 4-input multiplexer within each pixel. This multiplexer then, selects one input from the four outputs of the surrounding pixels, left, right, top or bottom. See Figure 4-9, which illustrates the 4-input multiplexer. Table 4-1 displays the direction selection table for each 4-input multiplexer and the overall array data flow direction.

Table 4-1: IP Mux Select Direction

			Data Flow
C_RC	C_LR	Mux_Out	Direction
0	0	From Left	Right
0	1	From Right	Left
1	0	From Up	Down
1	1	From Down	Up

The reduction in interconnection routing complexity generally limits the application of this system to algorithms that are orthogonal in nature, requiring only immediate neighbouring pixel values. Taking this into account, a motion compensation, wavelet transform and quantisation scheme, which is suited to this architecture is presented in [24] [44]. However, to perform the Zerotree coding algorithm a modified interconnection scheme becomes necessary, and is covered in Chapter 5. Algorithms such as arithmetic coding, which are predominantly sequential and require more versatile interconnections are typically not suited to this environment and as such conducted outside of the pixel array.

4.6. Array Mapping of Video Codec

Chapter 2 and Chapter 3 introduce several components relevant to different video coding schemes. This section proposes a novel video coding algorithm suited for implementation within an Intelligent Pixel Array Processor. Figure 4-15 illustrates a block diagram of the proposed codec for the IPA Processor. The extra dotted Zerotree component is included for the case where EZW is used over ZTE (See Chapter 5 for more information). Furthermore, two areas on the diagram have been defined, one for implementation within the array and the other for implementation outside the array, yet located on the same chip.

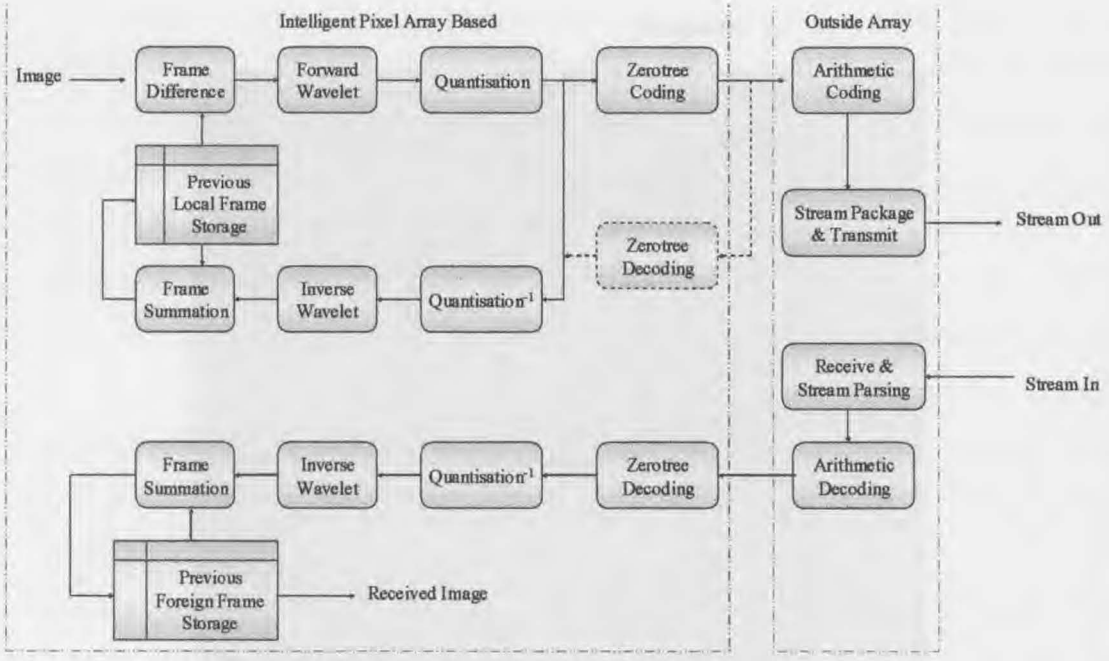


Figure 4-15: Proposed IPA Video Codec

The proposed System-On-Chip (SOC) view is presented in Figure 4-16. The main features of the proposed SOC include

- Capture, processing & display
- 25 frames per second PAL QCIF video compression
- Frame differencing motion compensation
- 3-scale triangular wavelet image decompositions
- Subband based quantisation
- ZTE or EZW entropy compression
- Arithmetic coding
- Stream assembly/disassembly for transmission
- Control circuitry
- Low clocking frequencies (100~200 kHz) to minimise power usage
- Integrated audio coder (Research conducted at the university of Las Palmas)

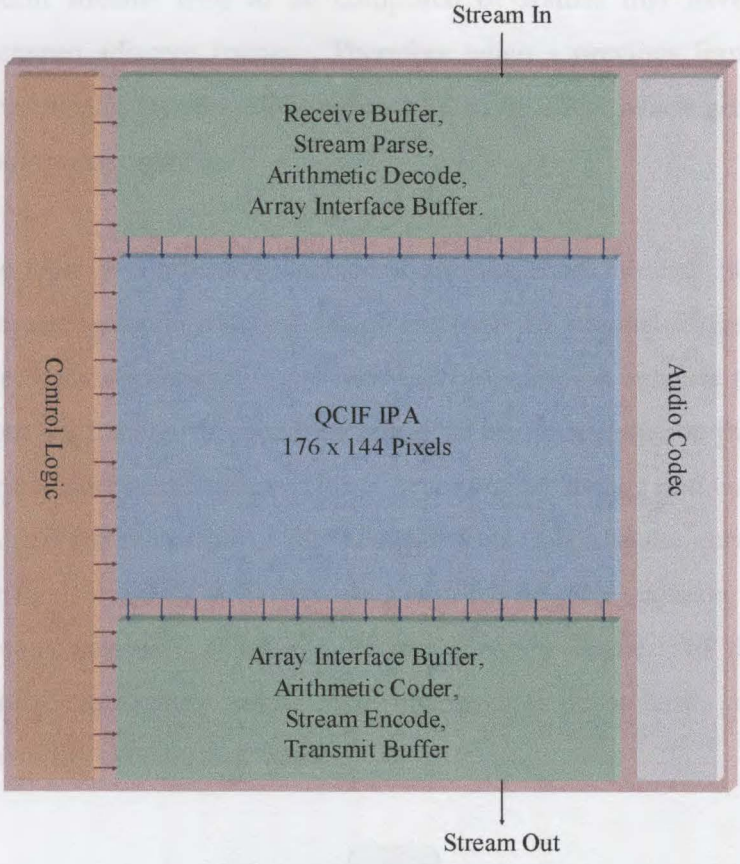


Figure 4-16: System-On-Chip Architecture

4.6.1. Motion Compensation Codec

The proposed IPA motion compensation codec employs frame differencing to minimise temporal redundancy between frames. Although, this technique is not as sophisticated as the block matching motion estimation (BMME) technique, it represents a trade-off between hardware complexity and the provision of minimal motion compensation ability. Techniques such as the BMME and other more complex motion compensation systems require the use of several of extra registers and components (or external RAM) on a per-pixel basis. Unfortunately, when employing a 0.25 micron CMOS technology, the required extra space is considered impractical especially for a massively parallel QCIF sized array. These complex techniques are better implemented on systems with large memory reserves.

A block diagram illustrating the operation of proposed frame differencing/summation technique is presented in Figure 4-17. This technique exploits the trait that most video

communication streams tend to be composed of frames that have large areas of similarity between adjacent frames. Therefore when a previous frame is subtracted from the current, the resulting differences tend to be small which generally improves the compression performance.

Quantisation typically introduces some error into the entire system. Since the decoder only receives the 'error introduced' image and not the original image at the sending device, it becomes necessary for the motion compensation scheme to take this into account when subtracting the previous image. This is because the previous image at the decoder contains some errors. This is overcome by storing and using the contents of the quantised frame instead of the 'raw' previous frame, at the encoder. Since this quantised frame is available at the decoding end, the decoder sums the incoming frame to the previous quantised frame to generate the new frame. In this manner, the decoded image somewhat 'tracks' the quantisation error and generates frames consistent with the encoder, for display.

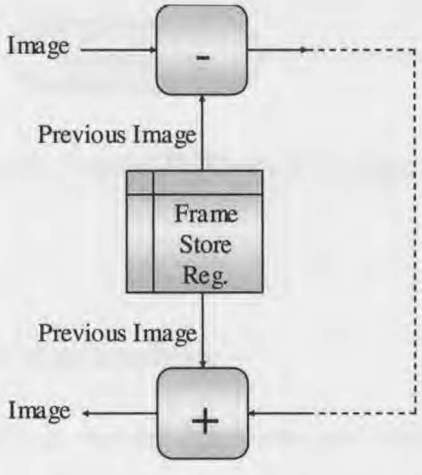


Figure 4-17: Frame Difference & Summation

With all pixels performing this process in parallel a frame difference of the entire image can be accomplished in 8~10 clock cycles, depending on the data precision chosen.

Frame differencing is easily performed within the proposed massively parallel hardware architecture, by the use of two storage registers, selected to match the precision of the input image, and one higher precision accumulator register, all distributed on a per-pixel basis. Each pixel also requires an arithmetic unit capable of performing both addition and subtraction. Since serial adders are smaller in size and subtraction is

equivalent to negative addition, one such serial adder is employed on a per-pixel basis. One of the two storage registers is employed to store the previous quantised frame content of the locally captured image, while the other is used to store the previous quantised frame content of the foreign frame, hence facilitating both encode and decode. Figure 4-18 illustrates the architecture required by each pixel to perform frame differencing and summation. In this case the system is configured to perform summation of a recently decoded value with the previous foreign frame to develop a new foreign previous frame for storage. Two other modes that can be performed include the ability to store a summated local frame, and to provide the motion difference to the accumulator register.

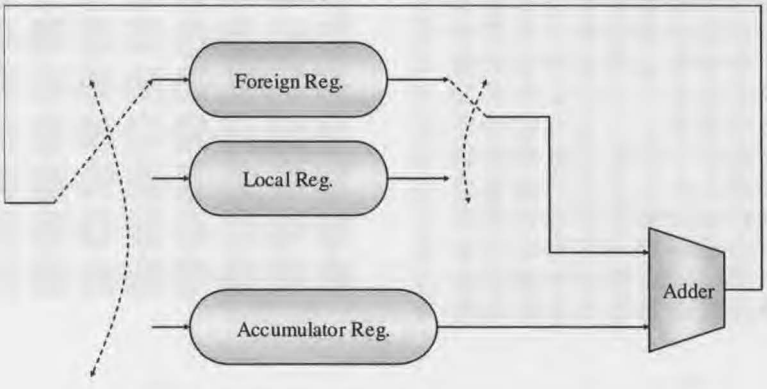


Figure 4-18: Frame Difference Components

4.6.2. Scale Control Architecture

For various purposes, such as wavelet transform and zerotree data load/extraction processes, a need exists to establish a scale based pixel enabling mechanism. Such a feature allows for the enablement of all pixels belonging to exactly one scale or sub-band throughout the whole array. The novel technique proposed here is heavily dependant on the positioning of sub-band related coefficients within a nucleic block, and involves the activation of a pixel via intersecting horizontal/vertical control lines. The design for an N x N nucleic block requires N horizontal (*HEnable*) control lines, with each horizontal line propagating to all pixels in a row, and N vertical lines (*VEnable*), each propagating to all pixels in a column. Figure 4-19 (A) shows the required *HEnable* and *VEnable* signals for a 3-scale nucleic block. Figure 4-19 (B) also shows the required control signals for a 16 x 16 pixel array, which consists of four 3-

scale nucleic blocks. It is observed that the same number of external control lines (8 + 8 = 16) can be used to drive the *HEnable* and *VEnable* lines in both figures. Similarly, given a 3-scale transform, any arbitrary sized array may be controlled via 8 vertical and 8 horizontal control lines. Generally the number of scales used (n) is related to the number of external control lines required by, 2^n for the horizontal and 2^n for the vertical, resulting in a total of 2×2^n control lines.

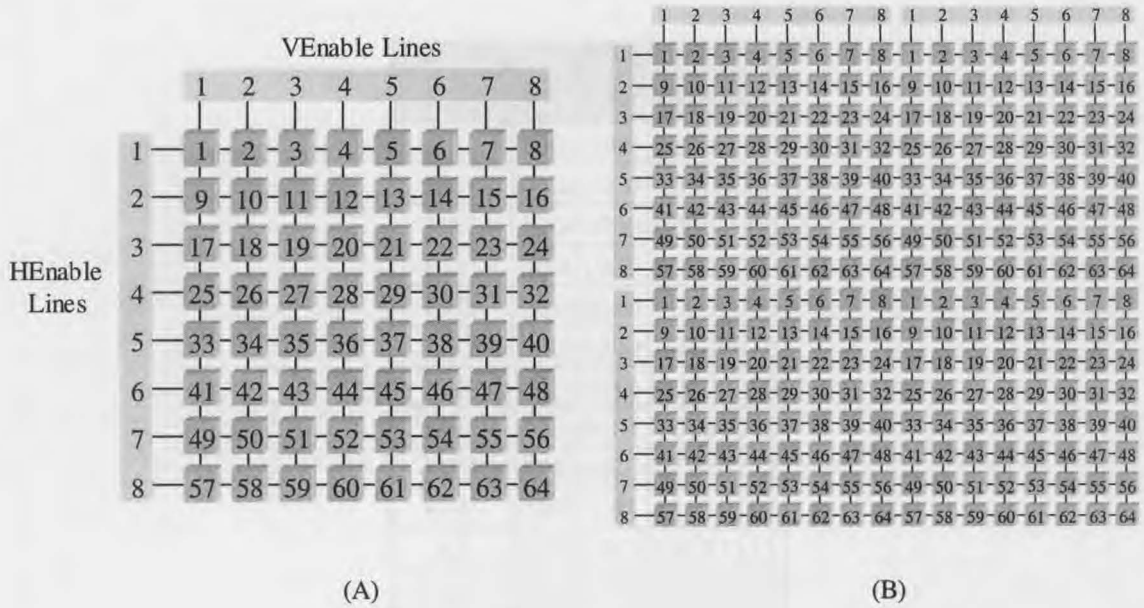


Figure 4-19: VEnable & HEnable Control Lines

In order to select a subband the relevant *VEnable* and *HEnable* signals are set high. Figure 4-20 illustrates the principal behind subband selection (HH_3) through activation combinations of *VEnable* and *HEnable*.

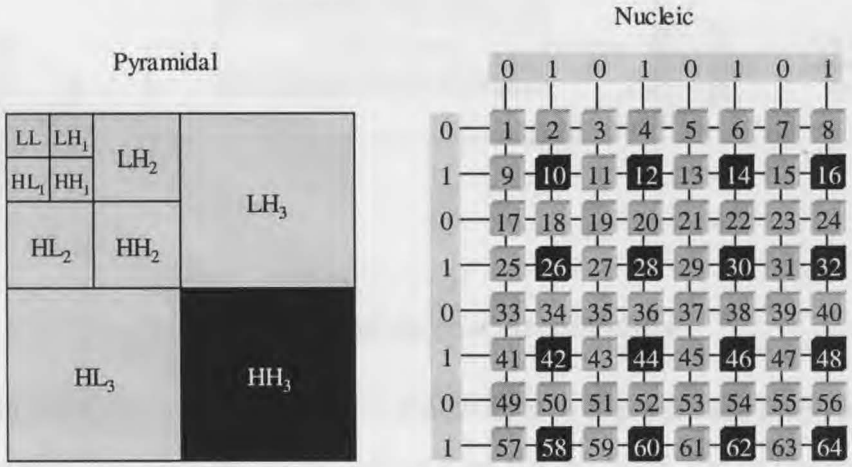


Figure 4-20: Activation of HH_3 subband via VEnable & HEnable

A full list of useful *VEnable / HEnable* activation patterns for a 3-scale nucleic block based array, of any arbitrary size, is presented in Table 4-2. Using this, specific subbands in a particular scale can be operated on. The last four entries in the table relate to wavelet transform operations, where more than one subband is active at one time.

Table 4-2: VEnable & HEnable list for subband activation.

Subband	VEnable (Bin)	HEnable (Bin)
LL	10000000	10000000
LH1	00001000	10000000
HL1	10000000	00001000
HH1	00001000	00001000
LH2	00100010	10001000
HL2	10001000	00100010
HH2	00100010	00100010
LH3	01010101	10101010
HL3	10101010	01010101
HH3	01010101	01010101
All	11111111	11111111
0 to 2	10101010	10101010
0 to 1	10001000	10001000
0	10000000	10000000

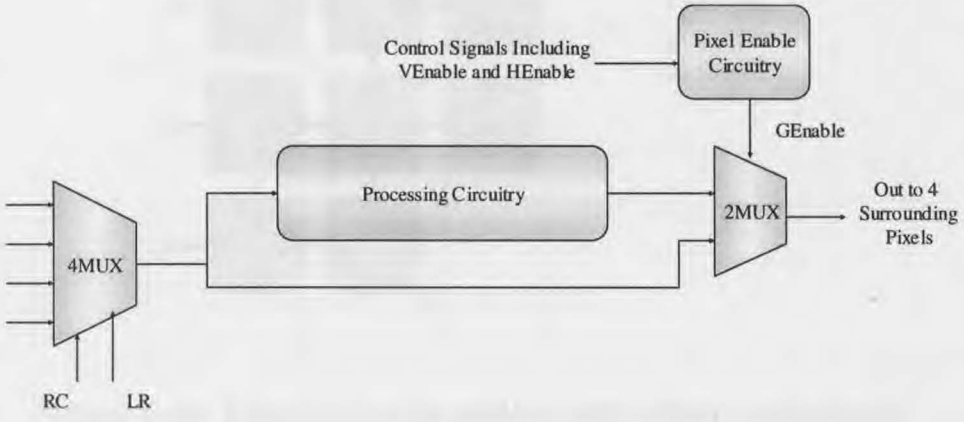


Figure 4-21: Pixel Bypass Architecture

A disabled pixel, (i.e. when *HEnable / VEnable* lines to that pixel are low) functions as a pass through unit for data. If data arrives from the left, right, top or bottom the data is repeated at the right, left, bottom or top respectively. The bypass architecture is

presented in Figure 4-21. In this mode nearly all functions of the pixel are disabled. The internal signal *GEnable* is employed to indicate the activation status of a pixel; if *GEnable* is logic high then the pixel is enabled otherwise disabled. *GEnable* is derived directly from *VEnable*, *HEnable* and other zerotree components described in Chapter 5. If either *VEnable* or *HEnable* is set low, then *GEanble* is forced low.

4.6.3. High Pass / Low Pass Pixel Identification

In order to perform the wavelet transform, each pixel requires information identifying it as a high-pass or low-pass pixel. To accommodate this each pixel is equipped with two pairs of signals. Signals *HPRin* and *HPRout* are used for row-wise identification, while signals *HPCin* and *HPCout* are used for column-wise identification. These signals cascade from one pixel to the other in either a row or column direction. Figure 4-22 illustrates the connection pattern for the first few pixels in an array. Logic zero is asserted at the top and left edges to initialise the high/low pass identification process. The output signals for the row and column are derived from the input signals based on the activation status of each pixel. Both outputs reproduce the resultants of switchable inverters, which invert the input signals in enabled pixels. Inactive pixels do not invert and reproduce the input signal at the output.

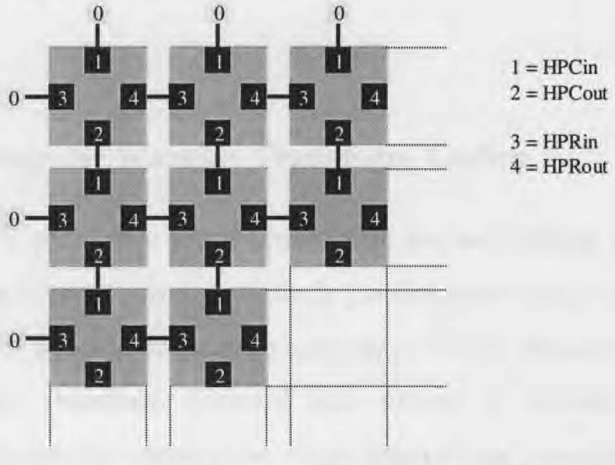


Figure 4-22: Low/High pass pixel identification architecture

If all pixels are enabled (i.e. scale 0-2), the array becomes a grid pattern of Low and High pass pixels. If every alternate pixel is enabled (i.e. scale 0-1), then the grid pattern is found in these alternate pixels. Figure 4-23 illustrates this pattern for when all pixels are activated and for when alternate pixels are activated. The first of the two-digit

number in each pixel indicates the column wise inverting pattern, while the second digit indicates the row-wise inverting pattern.

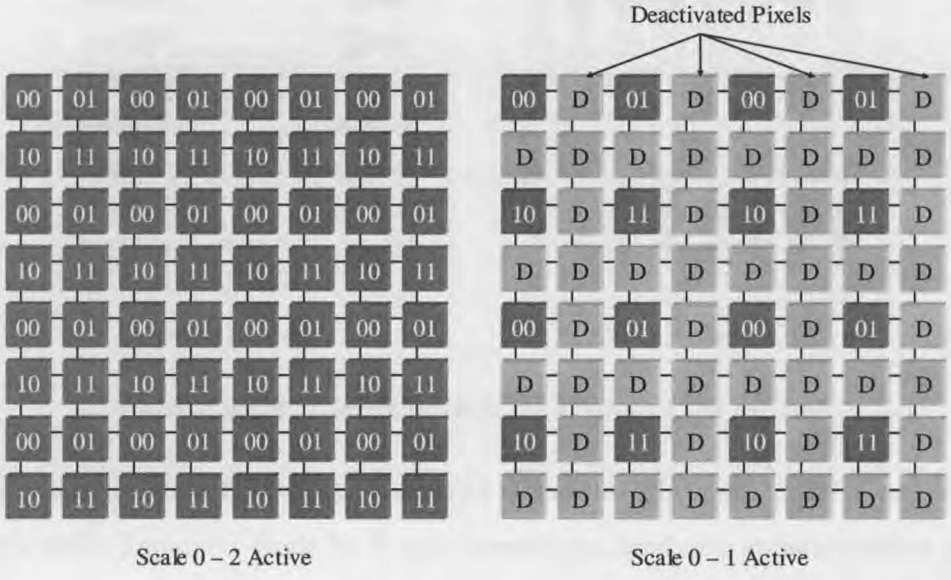


Figure 4-23: High / Low pass grid pattern

If a selection of the transform direction (Row / Column) is chosen, then technique provides pixels with High/Low pass identification. Although presented for one nucleic block of a 3-scale transform this holds true for all image sizes and nucleic block sizes.

4.6.4. Triangular Wavelet Transform Codec

The thesis in [44] presents a novel architecture for performing the triangular wavelet transform and its inverse, given a massively parallel processing architecture such as the IPA, unlike in [45] which represent the majority of SIMD implementations. Since a 2D triangular wavelet transform (forward and inverse) is orthogonal in nature, it is generally implemented by realising two single dimensional transforms, one for the rows and the other for the columns. The IPA implementation of the transform exploits this to perform the 2D full image transform in a row/column manner as illustrated in Figure 4-24. However, when performing either the row or column transforms, they are conducted in parallel for the entire array reducing the required clock cycles to around 150~200 for a 3-scale full image transform.

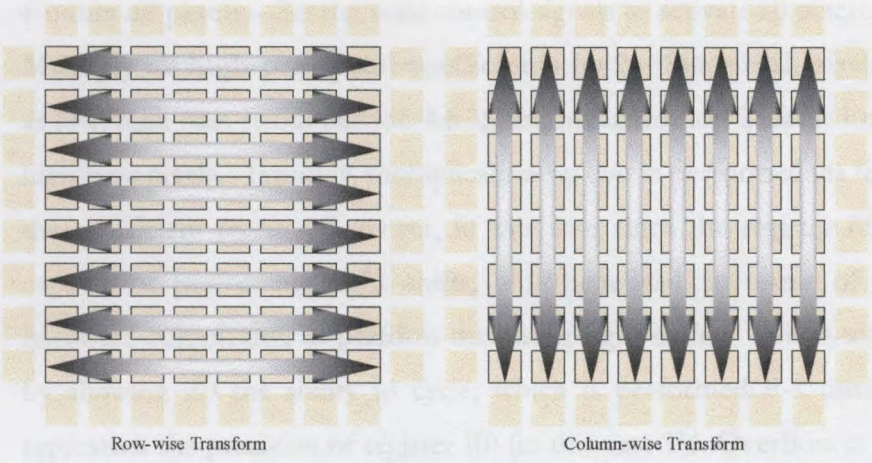


Figure 4-24: Row/Column 1D Transforms for 2D

Furthermore, since the triangular wavelet algorithm (Section 2.5.2) is realised with simple shifts (multiply/divide by 2) and summations, hardware implementation is easily performed via a single shift register, an arithmetic unit, some pixel enable logic and a high/low-pass selector on a per-pixel basis. The shift register provides for coefficient storage and multiplication. The arithmetic unit provides an addition/subtraction facility. The pixel enable logic allows for scale selection while the high/low-pass selector determines which pixels belong to a scale's high-pass/low-pass coefficient components. Figure 4-25 depicts the per-pixel architecture required to perform a multi-scale wavelet transform and inverse.

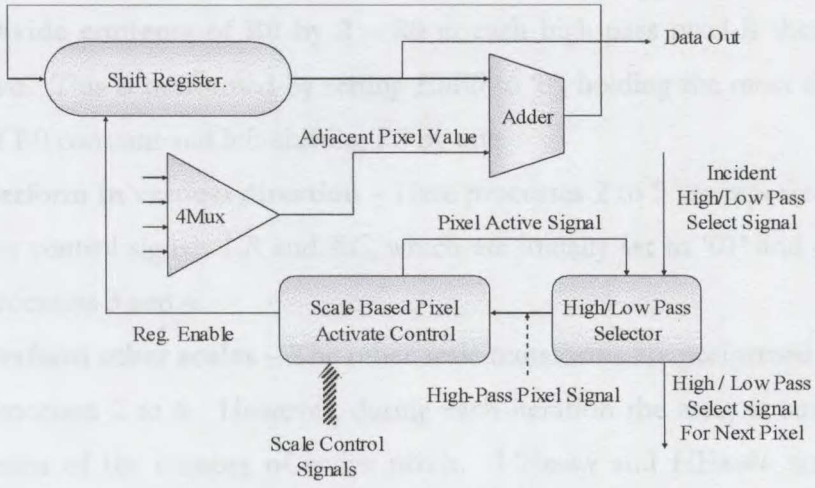


Figure 4-25: Wavelet Transform Architecture

The control structure for a forward transform as taken from Section 2.5.2, is as follows.

1. **Enable all pixels** – Set the scale control signals to activate all pixels.
2. **Multiply all high-pass pixel coefficients by 2** – Since each pixel is assigned as low/high pass (from Section 4.6.3), the multiplication is easily limited to the high-pass pixels. Typically, multiplication by two is performed via left shifts on the coefficient register, however, in this case since the register R0 is already capable of performing right shifts, it is beneficial, in terms of minimising hardware complexity, to perform this using right shifts. This is accomplished by allowing R0 the ability to cycle, which is performed P-1 times, where P represents the precision of register R0 (in this case 9). Overflow is a non-issue if the chosen precision for the wavelet coefficient is limited to P-3 (this case 6) bits and 2's complement numbering is maintained.
3. **Subtract low-pass coefficient from left** – Register R0, in all low-pass pixels, during the wavelet transform is fixed to rotate. In comparison the high-pass pixels are routed to receive data from the accumulator component. The control lines *LR* and *RC* are first set to '00' to accept data from the left. The accumulator module is then forced into subtract mode via setting a control signal *S* to '0', and a carry bit is preset. The clock into R0 is then enabled via the control signal *EnR0*. Nine clock cycles are then applied to attain the subtracted value in R0 in all high-pass pixels.
4. **Subtract low-pass coefficient from right** – This is similar to the previous stage, with the exception that, *LR* and *RC* are now set to '10'.
5. **Divide contents of R0 by 2** – R0 in each high-pass pixel is then divided by two. This is performed by setting *EnR0* to '0', holding the most significant bit of R0 constant and left-shifting R0 by one.
6. **Perform in vertical direction** – Here processes 2 to 5 are repeated, except for the control signals *LR* and *RC*, which are initially set to '01' and then '11' for processes 3 and 4.
7. **Perform other scales** – The other scale transforms are performed by repeating processes 2 to 6. However, during each iteration the array is sub-sampled in terms of the number of active pixels. *VEnable* and *HEnable* are established according to Table 4-2 to select the low-pass pixels from a previous stage.

The forward wavelet transform results in the generation of a multiple nucleic blocks throughout the array. In the case of a 32 x 32 pixel array, using a three scale transform,

4 x 4 nucleic blocks of size 8 x 8 pixels are generated. During the inverse transform the image is developed across the array, instead. The inverse transform is performed by repeating processes 1 to 7, with the exception that additions are performed instead of subtractions in processes 3 and 4.

4.6.5. Coefficient Quantisation and Inverse

Quantisation is only a necessity for coefficients used in the zerotree entropy coder (ZTE), because the embedded zerotree wavelet (EZW) algorithm harbours an embedded quantisation mechanism within its coding algorithm. The ZTE quantisation mechanism is simply implemented by a sub-band based coefficient truncation technique. This generally limits the number of coefficient quantisation levels possible integer multiples of two. However, when considering the reduction in per-pixel circuitry posed by a simple right-shift truncation, in comparison to division, the trade-off is easily justified. The proposed quantisation architecture is presented in Figure 4-26.

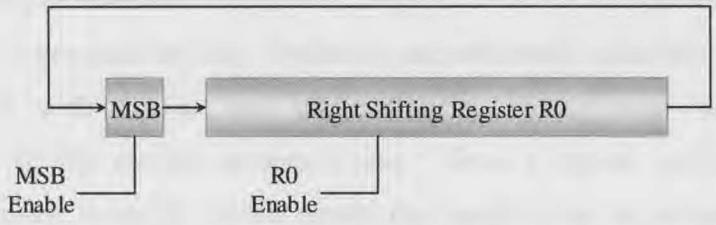


Figure 4-26: Quantisation Architecture

The quantisation process relies on the rightward bit shift of coefficients in register R0. However, to perform correct 2's complement quantisation the MSB bit is held constant by disabling the clocking of the MSB. By enforcing that register R0 rotates only when a pixel is active, entire sub-bands can be made active and quantised to a particular level. The number of bits truncated (T), when quantising to a particular level, is directly proportional to the number of clock-cycles issued.

The inverse quantisation process is performed by reversing the direction of the shift register. However, since it is costly to implement a bidirectional shift register, the

inverse is performed via right-shifts. If the rotate feature of the register is enabled and MSB is also enabled to rotate, then performing *B-T* clock cycles, where *B* represents the number of bits in register R0 (in this case 9) and *T* represents the number of quantisation clock cycles, results in the inverse quantisation of the coefficient.

Examining an example; if R0 contained 0 01001011b (75) and if two bits are truncated the resultant becomes, 0 00100101b (37) after the first clock cycle, and 0 00010010b (18) after the second. During the inverse phase the register is cycled seven (9-2) times in the following manner from 0 00001001b to 1 00000100b to 0 10000010b to 0 01000001b to 1 00100000b to 0 10010000b and to finally result in 0 01001000b (72).

Once the quantisation is completed the ZTE encode stage takes place, as covered in Chapter 5.

4.6.6. Stream Codec

The zerotree coding stage precedes the arithmetic coding stage. Design of an arithmetic coding architecture is considered beyond the scope of this thesis; however, a suitable design is presented in [64]. Optimally, the arithmetic coder should be designed to be adaptive, with at least two statistical tables to represent the symbol and coefficient set of the chosen zerotree coder. Since a typical arithmetic coder is sequential in nature, it may be placed outside the parallel array yet remain on the same chip. Once arithmetic coding has been performed the stream is ready to be packed and transmitted. For ideal performance at least two FIFO based I/O buffers are required to maintain the data rates. Conventional buffer designs such as those generated via the XILINX buffer generator may be used.

4.7. *Primary Component Schematics*

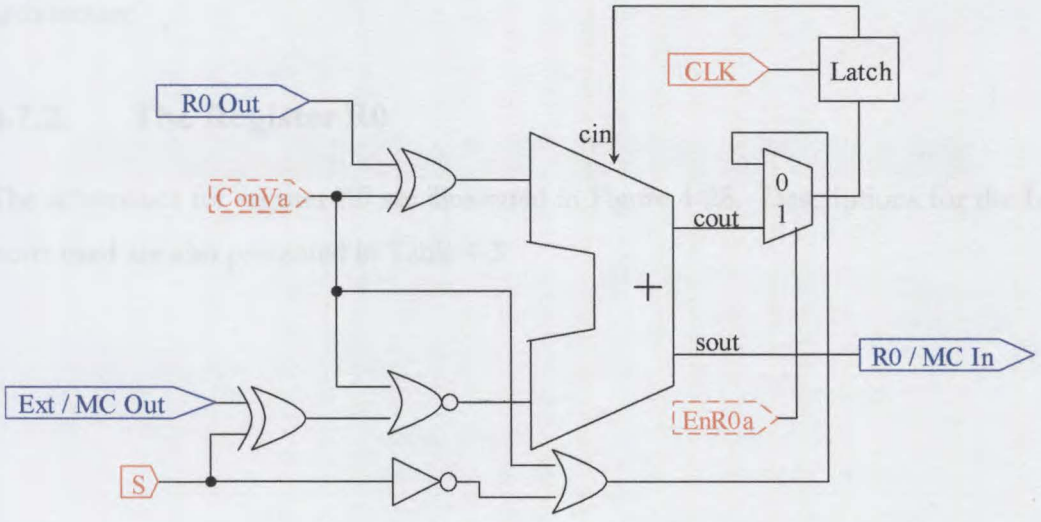
This section presents a series of components in schematic form, necessary for the implementation of the proposed wavelet based zerotree codec. Components presented herein are designed to integrate to either, the EZW or the ZTE codecs. As such optimisations for a particular implementation have been omitted. The designs have been

verified in VHDL and have been shown to adequately perform the required tasks. Tools such as Synopsys, Cadence and VHDLSimili have been employed for the verification of these architectures.

The designs contain two types of I/O representations, I/O ports drawn with either continuous or dashed lines. The latter represents signals generated and used within individual pixels, while the former indicates control signals originating externally to pixels. Furthermore blue I/O ports represent data compared to the red I/O ports which represent control lines.

4.7.1. Summation Unit

The schematic representation of the summation unit or accumulator is presented in Figure 4-27.



ConVa	S	EnR0a	Mode	Clk Cycles
0	0	0	Carry Set	1
0	0	1	Subtract	Data Dep.
0	1	0	Carry Clear	1
0	1	1	Add	Data Dep.
1	0	0	Carry Set	1
1	0	1	Convert	Data Dep.
1	1	0	Carry Clear	1
1	1	1	Convert	Data Dep.

Figure 4-27: Summation Unit Schematic

The summation unit is capable of performing the following tasks.

1. **Bitwise Addition** – This is performed by firstly, clearing the latch, then setting control signal *S* to high and clocking for one clock cycle. Once *EnR0a* is set high, bit-wise data arriving at *EXT/MCOUT* and *R0out* are summated to produce a bit-wise result at *R0/MCin* when clocked.
2. **Bitwise Difference** – This is performed by firstly, setting the latch, then setting control signal *S* to low and clocking for one clock cycle. This sets up the circuit to perform 2's complement differencing or subtraction by negative addition. Once *EnR0a* is set high and the circuit is clocked data from the two inputs are differenced according to *R0/MCin = R0out + (-EXT/MCOUT)*.
3. **Bitwise 2's Complement Conversion** – This mode is used to convert the contents arriving at *R0out* into 2's complement mode and vice versa. For this to occur *S* must be set low for one clock cycle in order to set the latch. Then both *EnR0a* and *ConVa* are set high and the circuit is clocked.

EnR0a and *ConVa* are both derived signals, which are activated by other parts of the architecture.

4.7.2. The Register R0

The schematics for register R0 are illustrated in Figure 4-28. Descriptions for the I/O ports used are also presented in Table 4-3.

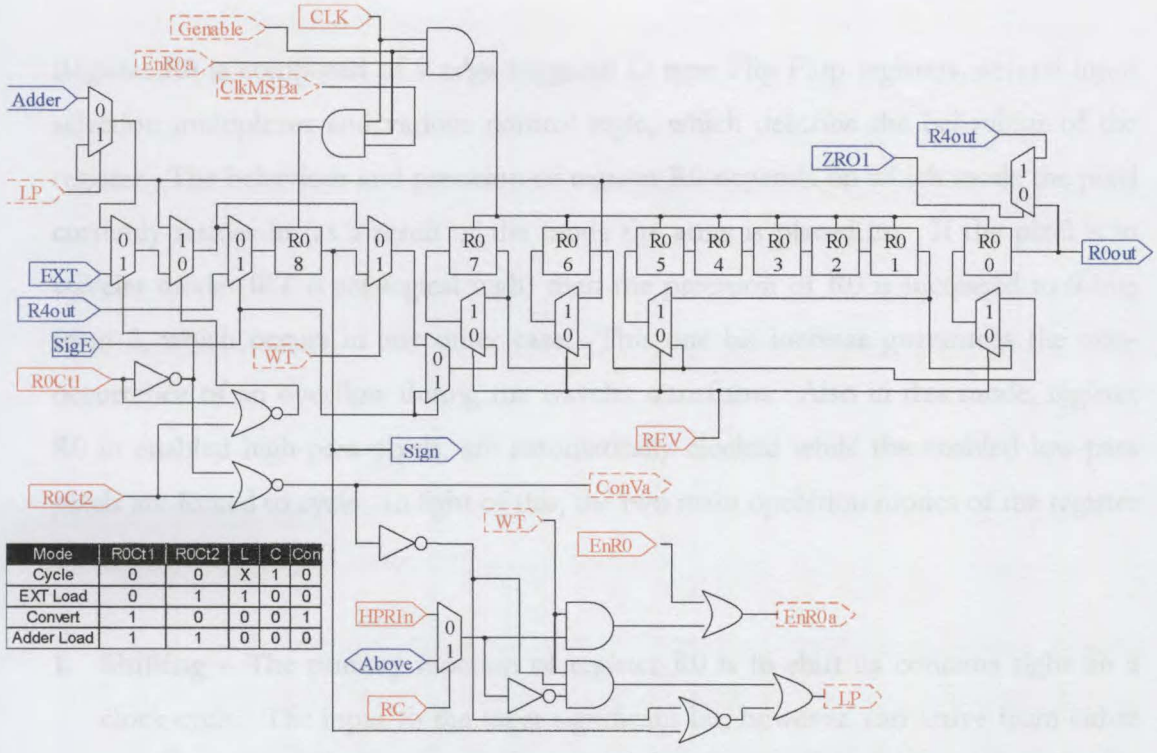


Figure 4-28: Register Ro Schematic

Table 4-3: Register Ro I/O Ports

Signal	Dir	Type	Origin	Description
CLK	In	Clock	External	Clock signal
EnR0	In	Control	External	R0 Clock Enable
HPRin	In	Control	External	Row High-Pass Selector Input
R0Ct1	In	Control	External	Mode Select 1
R0Ct2	In	Control	External	Mode Select 2
RC	In	Control	External	(Row) Column Selector
REV	In	Control	External	Reverse R0
ClkMSBa	In	Control	Internal	R0 MSB Clock Enable
ConVa	Out	Control	Internal	Generated Convert Signal
EnR0a	Out	Control	Internal	Generated Pass Based R0 Enable
GEnable	In	Control	Internal	Pixel Enabled Signal
LP	Local	Control	Internal	Low-Pass Identification
WT	In	Control	Internal	Wavelet Transform Mode Select
Above	In/Out	Data	External	Column High-Pass Selector Input
Adder	In	Data	Internal	Data From Summation Unit
EXT	In	Data	External	Data From Other Pixels
R0out	Out	Data	Internal	Data To Other Components
R4out	In	Data	Internal	Data From Zerotree
R5out	In	Data	Internal	Data From Zerotree
SigE	In	Data	Internal	Data From Zerotree ID
Sign	Out	Data	Internal	Data to Zerotree
ZRO1	In	Data	Internal	Data from Zerotree

Register R0 is composed of 9 edge-triggered D type Flip Flop registers, several input selection multiplexes and various control logic, which describe the behaviour of the register. The behaviour and precision of register R0 depends on which mode the pixel currently resides in (as a result of the mode the array is placed in). If the pixel is in wavelet mode (*WT* is set logical high) then the precision of R0 is increased to 9-bits from 8, which occurs in any other case. This one bit increase guarantees the non-occurrence of an overflow during the wavelet transform. Also in this mode, register R0 in enabled high-pass pixels, are automatically clocked while the enabled low-pass pixels are forced to cycle. In light of this, the two main operation modes of the register R0 are as follows.

1. **Shifting** – The primary function of register R0 is to shift its contents right on a clock cycle. The input to the most significant bit, however, can arrive from either another pixel (*EXT*), the summation unit (*Adder*), LSB of R0 (*R0out*) or the zerotree components (*SigE*, *R4out* or *R5out*). The significant amount of control is used to facilitate the multiple sources and some other features used for different operations in other processing stages.
2. **Reverse** – When the *Rev* signal is set high, the register swaps R0_0 and R0_1 with R0_6 and R0_7 and performs a cycling function all within one clock cycle. By applying one swap for ever four shifts the contents of register R0 is reversed. Table 4-4 shows an example of how the contents of R0 are reversed within nine clock cycles. The reversing function is used for the EZW codec and is described further in Chapter 5. The reverse function should only be activated in pixel shift (*SH* = 1) mode.

Table 4-4: Ro Reversing Function

	Register R0							
Rev	7	6	5	4	3	2	1	0
1	1	2	3	4	5	6	7	8
0	1	8	7	3	4	5	6	2
0	2	1	8	7	3	4	5	6
0	6	2	1	8	7	3	4	5
1	5	6	2	1	8	7	3	4
0	5	4	3	2	1	8	7	6
0	6	5	4	3	2	1	8	7
0	7	6	5	4	3	2	1	8
0	8	7	6	5	4	3	2	1

In addition to these operating modes register R0 contains other features to accommodate various requirements requested by different components in the processing stage. The features include the ability to aid in the conversion of 2's complement value into sign magnitude representation (via *R0Ct1* and *R0Ct2*), hold the most significant bit (R0_8, Sign bit) constant (via *ClkMSBa*), behave differently depending on high/low pass pixel status (via *LP* and *EnR0*) and extract/load sign/magnitude values from the zerotree components (via *SigE*, *R0out*, *R4out*, *R5out* and *ZRO1*).

4.7.3. Motion Compensation Unit

The motion compensation unit is based on the storage of two 6-bit pixel values and the redirection of these values to different processing components. The schematics for the motion compensation unit are provided in Figure 4-29. It can accept data from two sources *R0out* (register R0) or *Adder* (accumulator unit) and produces resultant data via *MCout*. The two registers, R1 and R2 are both composed of edge-triggered D-type flip-flops (DFFs) interconnected in a manner to allow right shifts of data. In this implementation, the previous frames (foreign and local) are represented by 6-bit pixel values; therefore, these registers are designed to contain 6-bit values (i.e. 6 DFFs).

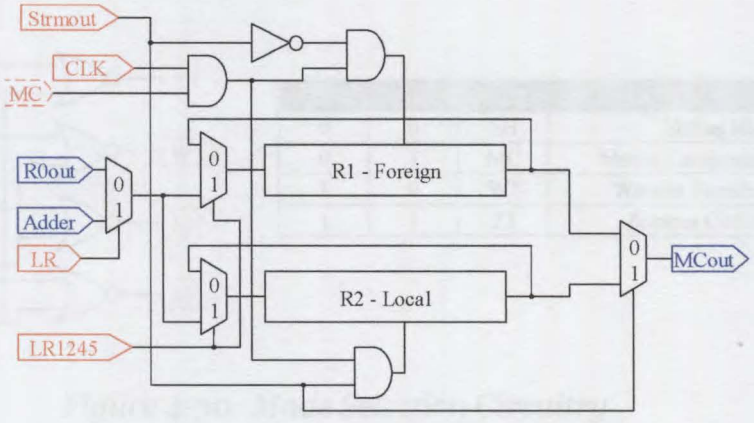


Figure 4-29: Motion Compensation Unit

Table 4-5 lists all signals used in the motion compensation unit and their respective descriptions. The signal *LR* has been reused to select between the inputs *R0out* and *Adder*, as the pixel interconnection direction is not required during this simple motion compensation stage. For any clocking to occur in motion compensation mode, the

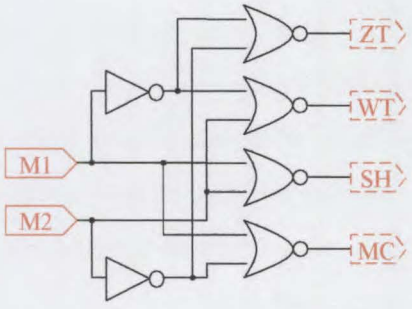
internal control signal *MC* must be held high. The signal *Strmout* determines which of R1 (previous foreign pixel storage) or R2 (previous local pixel storage) is activated. R1 is activated when *Strmout* is low, else R2.

Table 4-5: Motion Compensation Signals

Signal	Type	Description
CLK	Clock	Clock Signal
LR	Control	Selects Load from R0 or Adder
LR1245	Control	Selects Register Load/Rotate Mode
Strmout	Control	Selects R1 or R2 Clocking and Output
MC	Control	Activates Motion Compensation Unit

4.7.4. Mode Selection Circuitry

In order to switch the pixel and ultimately the array into specific operation modes, special mode selection circuitry is required. The mode selection circuitry is primarily dependant on two global control signals M1 and M2, which propagate to all pixels. Depending on the values of M1 and M2 the pixel can operate in one of four modes, which includes Motion Compensation Mode (MC = 1), Wavelet Mode (WT = 1), Zerotree Mode (ZT = 1) and Data Shift Mode (SH = 1). Figure 4-30 illustrates the schematics for the proposed mode selection circuitry.



M1	M2	Mode	Description
0	0	SH	Shifting Mode
0	1	MC	Motion Compensation Mode
1	0	WT	Wavelet Transform Mode
1	1	ZT	Zerotree Codec Mode

Figure 4-30: Mode Selection Circuitry

The selected mode applies to all pixels in the array and enables special processing circuitry relevant to the particular mode.

4.8. Conclusion

In this chapter, the main design aspects of a novel massively parallel pixel-wise processing array with potential to capture, compress, decompress and display real-time video in a single 3D OPTO-VLSI device, has been explored. An optics plane, oriented perpendicularly to the processing circuitry and electronic interfacing, has been shown to interface high bandwidth data for video capture and display. In comparison, the 2D VLSI processing plane has been shown to interface low bandwidth compressed data to and from another such device (or relevant software). The Intelligent Processing Array (IPA) has been shown to be composed of N x M (In the case of a QCIF image there are 176 x 144 pixels) processing elements (PEs) termed Intelligent Pixels (IPs). Each IP has the capability to perform light capture, via a photodiode/ADC, perform some processing, via VLSI circuitry, and display a pixel value, via a LCD based SLM. The VLSI circuitry, situated underneath the display, has been designed to perform frame differencing, triangular forward/inverse wavelet transform and sub-band based quantisation mechanisms, all on a per-pixel basis. Furthermore, these VLSI architectures also possess the ability to interface with an integrated zerotree codec, such as the EZW or ZTE architectures, presented in Chapter 5. This chapter fully details these architectures and associated processing components, which have been verified for functionality.

Chapter 5

MASSIVELY PARALLEL ZEROTREE CODEC FOR THE IPA

"The significant problems we face cannot be solved at the same level of thinking we were at when we created them."

Albert Einstein (1879-1955)

5.1. Introduction

Recent times have introduced a number of hardware designs/implementations of zerotree algorithms such as those described in Chapter 3. Some more prominent designs are presented in [65] [66] [67]. Essentially, these designs are based for operation on SIMD processors and as such a massively parallel approach has not as yet been investigated. The work presented in this chapter illustrates the design of a massively parallel self-classifying array of pixel-wise processing elements, which as a whole perform the duties of a zerotree algorithm for the purpose of image/video compression. Two zerotree codec designs have been presented, firstly the EZW and secondly the ZTE algorithm. A choice is then made for the selection of the most suited algorithm for VLSI implementation within the IP100P parallel processing array prototype. Tackled herein are problems relating to parallel implementations of hierarchical zerotree propagation, pixel based self-classification of zerotree symbols and coefficients, integration of a zerotree codec to the wavelet transform, parallel data load/extract for varying number of pixels and array-wise replication of

processors for arbitrary sized array generation. The architecture and schematics presented herein have been verified through VHDL simulations conducted in Synopsis and/or VHDLSimilli.

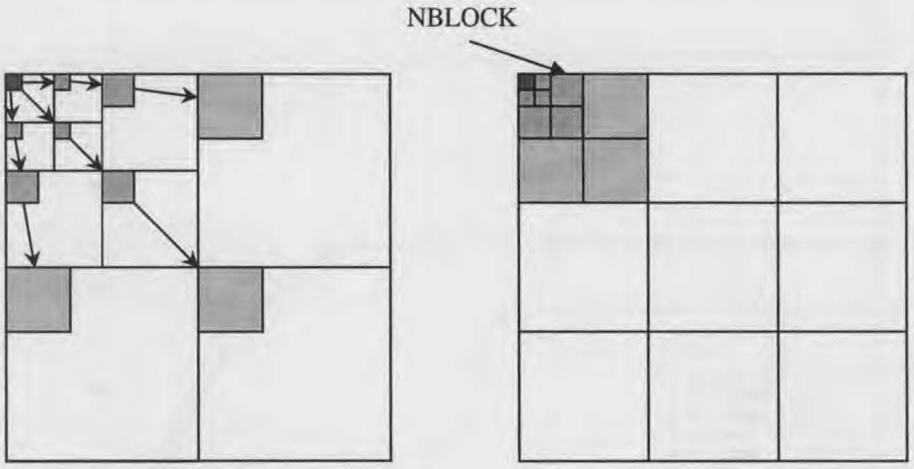
5.2. *Parallel Significance Tree Propagation*

An interesting challenge that surfaces when considering the design of VLSI architectures for zerotree coding, relates to the method chosen to implement the significance propagation tree between related parent and descendant pixels. As discussed in Chapter 3 zerotree algorithms require significance information sharing between parent and descendant pixels that belong to similar spatial positions but are located within the multiple scales of a wavelet transform. Typically, the wavelet coefficients are organised by scale into a memory bank, which is then traversed in multiple passes identifying significant coefficients and then coding relevant symbols for those coefficients. This technique is ideal for conventional SIMD processors. The organisation of the memory bank depends on the search technique selected, from a typical choice between Depth First Search and Breadth First Search, which are described in Chapter 3. To introduce an increase in processing speed, some solutions adopt a MISD approach, which subdivides the wavelet coefficient map into three distinct sets of data, to which individual processors are allocated to perform parallel processing on the main tree. Extending this concept significantly, this section proposes a parallel search scheme that allows pixel-wise search of the entire tree in a MIMD approach.

5.2.1. Nucleic Blocks and Scale Selection

As introduced in Chapter 2, a nucleic block (NBLOCK) consists of all related spatial coefficients resulting from the wavelet transform of an image. Figure 5-1 (a) shows the array-wise distribution of coefficients generated by a typical SISD wavelet transform technique, while Figure 5-1 (b) illustrates the NBLOCK equivalent. This indicates that an NBLOCK contains exactly one wavelet tree, and as such forms one zerotree dependence tree. Furthermore, it implies that each of these trees can be processed independently of each other and, therefore, in parallel. An NBLOCK is perfectly square entity with dimensions that are purely governed by the number of scales used in

the wavelet transform that generated it. If Sc represents the number of wavelet scales and the size of the NBLOCK is represented by $L = K^2$, then the dimension K is defined as $K = 2^{Sc}$. Therefore a three scale wavelet transform will generate a 8×8 NBLOCK.



(a) Image-wise tree dispersion (b) NBLOCK tree dispersion

Figure 5-1: NBLOCK Coefficient Tree Representation

The proposed parallel scheme is based on each pixel receiving the significance status of all its relations instantaneously and as they are determined. In order for this to occur, in parallel across all pixels in the array, fixed interconnections are required between the parents and their descendants. This is described in the next section. An effect that results is the enforcement of fixed sized NBLOCKs pertaining to a particular implementation, as the interconnections are hardwired and not dynamically allocated. This also imposes a restriction on the number of scale levels for the wavelet transform as this must comply with that chosen for the zerotree component. Literature suggests [22] that typical image compression algorithms employ 3 – 6 scale transforms. However, as the number of scales increase, so does the number of interconnections and routing complexity of those interconnections within an NBLOCK. Therefore, a compromise is required. Figure 5-2 illustrates a series of performance curves for different images coded at 64K. Each of the images is compressed with a 1 – 8 scale wavelet transform and zerotree coder. As is clearly seen, a 3 scale and an 8 scale codec show very little difference in compression performance. This suggests that a 3 scale system while providing the least complex interconnection pattern also provides adequate compression performance. Figure 5-3 extends this to display the effect of a varying bit-count, and shows that with the exception of very low bit-counts a 3 scale

system performs quite adequately. Although 3 scale systems are described hereafter, the principals apply for all other scaled systems, with the exception of the parallel interconnection pattern.

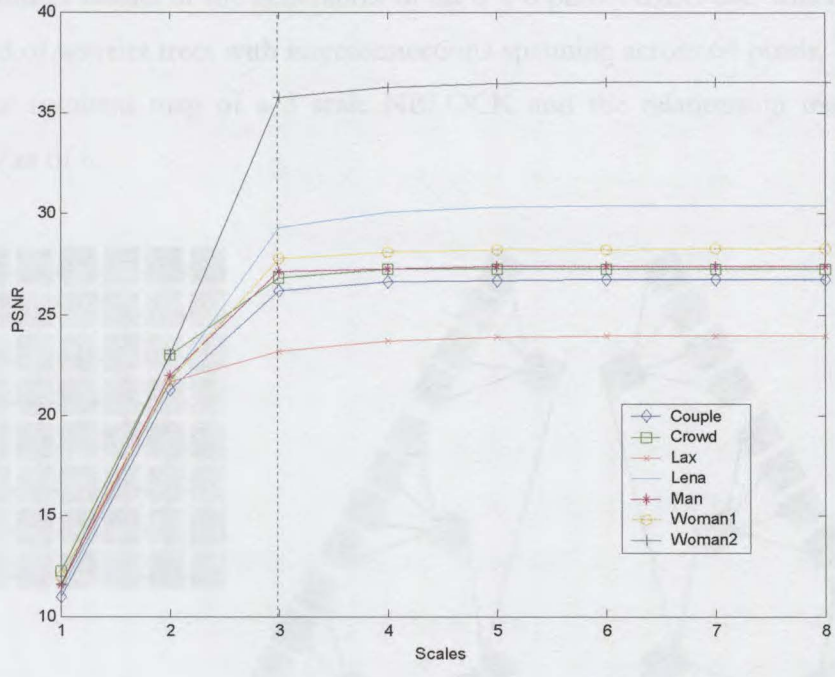


Figure 5-2: Scales vs. PSNR for Different Images

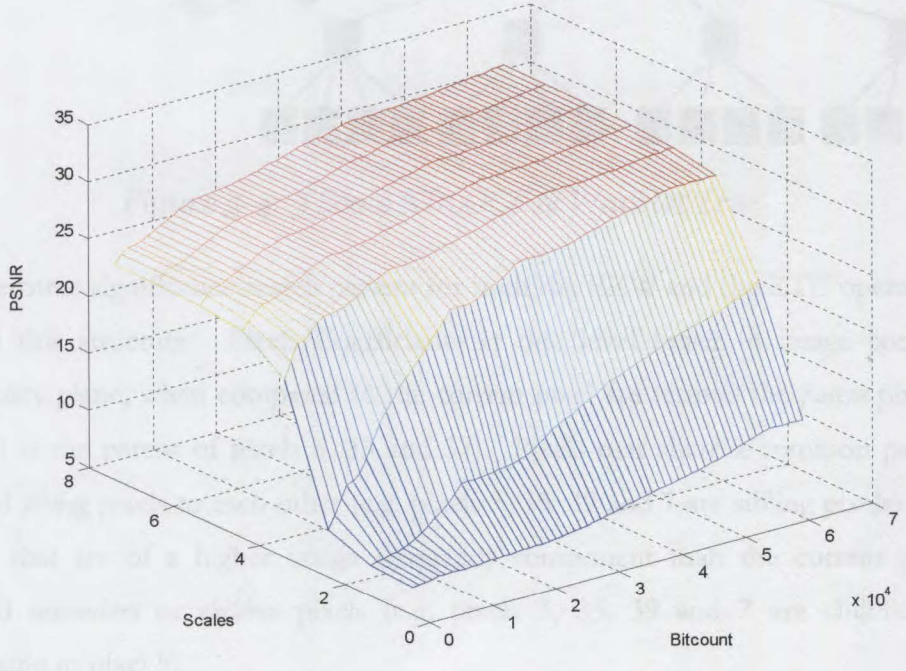


Figure 5-3: Performance for Varing Bit-counts & Scales

5.2.2. Tree Interconnections and Propagation

Once identified, the number of chosen scales (in this case 3) dictates the size and interconnection pattern of the NBLOCK. As previously discussed, a 3 scale decomposition results in the generation of an 8 x 8 pixel NBLOCK, which in turn is composed of wavelet trees with interconnections spanning across 64 pixels. Figure 5-4 shows the resultant map of a 3 scale NBLOCK and the relationship tree structure generated as of it.

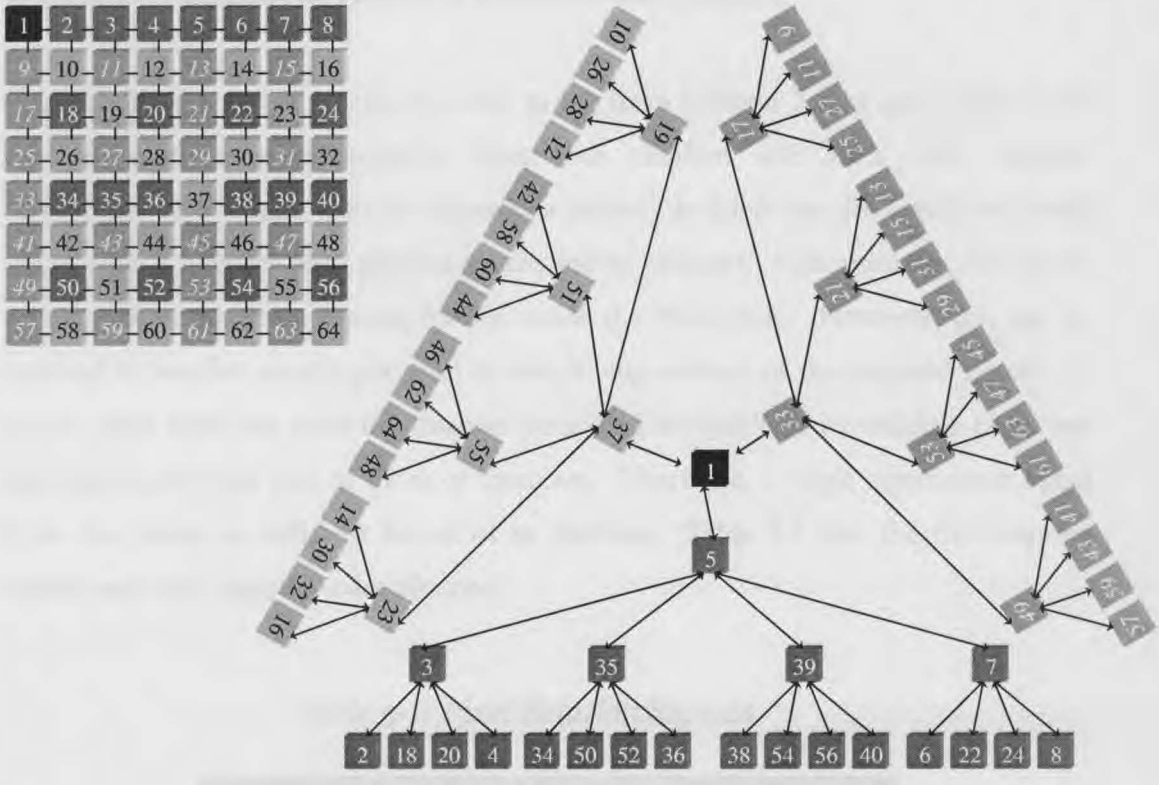


Figure 5-4: 3 Scale NBLOCK & Wavelet Tree

The zerotree significance search pattern for both the EZW and the ZTE operate solely within this structure. Pixels/Coefficients at one level lower in image component frequency plane, when compared to the current pixel are termed the *parent* pixels (e.g. pixel 1 is the parent of pixels 5, 33 and 37). Pixels that share a common parent are termed *sibling* pixels to each other (e.g. pixels 3, 35, 39 and 7 are sibling pixels). Finally pixels that are of a higher image frequency component than the current pixel are termed *descendant* or *children* pixels (e.g. pixels 3, 35, 39 and 7 are children pixels belonging to pixel 5).

Each NBLOCK always contains exactly one low-pass image component (pixel 1), several intermediate image components (e.g. pixels 3, 35, 39, 7 etc.), but is largely composed of high-pass image components at the fringes of the tree (e.g. pixels 2, 18, 20, 4 etc.). The single low-pass pixel has no parent pixels only descendant pixels therefore these pixel are always transmitted. The intermediate pixels contain both parent pixels and descendant pixels and therefore can contain different transmission status depending on both the parent and children pixel significance statuses. The high-pass pixels contain no descendant pixels and as such only depend on their current status and the significance statuses of their immediate parents.

Generalising this for all pixels, it is clear to see from Chapter 3 that each parent pixel requires significance information from four children and each child requires significance information from its immediate parent. In hardware this can be achieved via 10 signals between each pixel its parent and its children. Unfortunately, this places a heavy interconnection routing burden inside the NBLOCK. However, this can be reduced to just five signals per pixel by simplifying content of the required signals. A parent pixel does not need information pertaining to which of its children pixels are significant, only that one or more of them are. Therefore, a single significance signal from the parent is sufficient for all of its children. Table 5-1 lists the five required signals and their respective descriptions.

Table 5-1: Pixel Relation Signals

Signal	Dir	Description
Pin	IN	Significance indication from parent
Cin	IN	Significance indication from children
Sbin	IN	Significance indication from siblings
Pout	OUT	Significance indication to children
Sbout	OUT	Significance indication to sibling

The *Sbin* and *Sbout* signals exist as a result of the simplification process, where by significance data from one sibling is passed to the next in line and so on until the last sibling in the chain passes the amalgamated significance signal to the group's parent pixel. Figure 5-5 illustrates the five signal based interconnection pattern between the children and the parent pixels. Since the highest frequency component pixels have no descendants the *Cin* signals for those pixels are connected to logic zero. Also the first

pixel (e.g. 2, 3, 34, 38, 6 etc.) in a set of sibling pixels receives a zero in the *Sbin* signal as *Sbin* is a one way signal and the pixel contains no related sibling to the left of it.

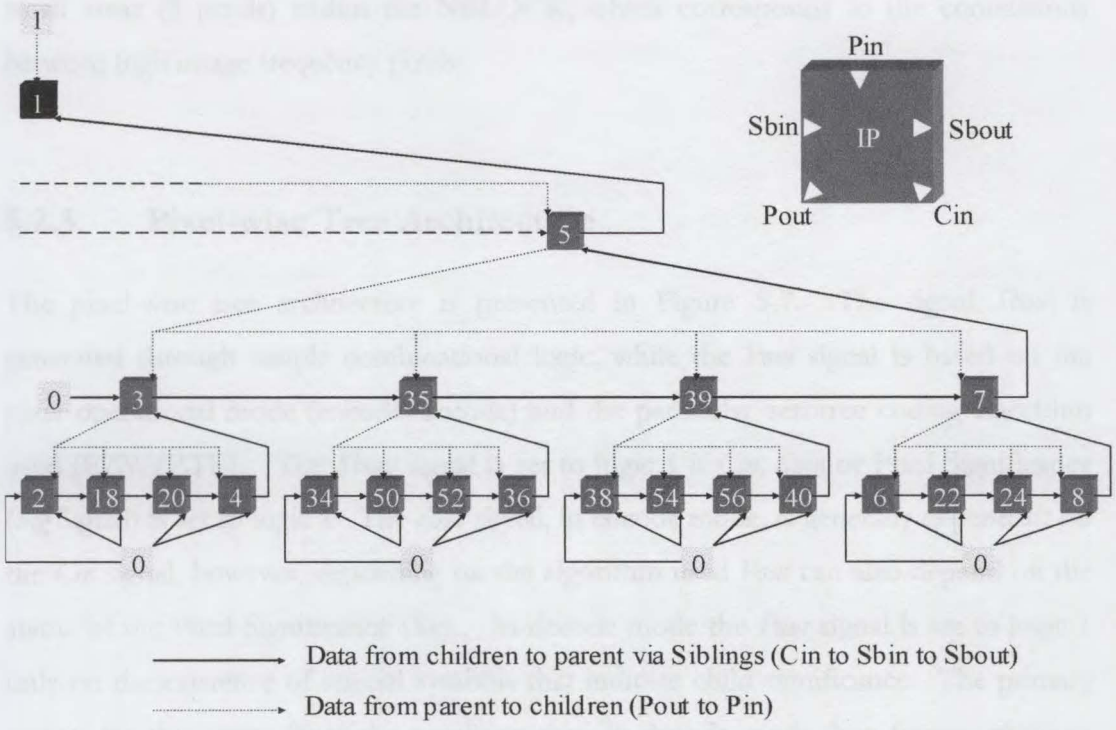


Figure 5-5: Single Branch of Five Signal Relation Tree

Pixel 1 receives logic 1 in its *Pin* signal as this indicates that these pixels have to be transmitted regardless of the state.

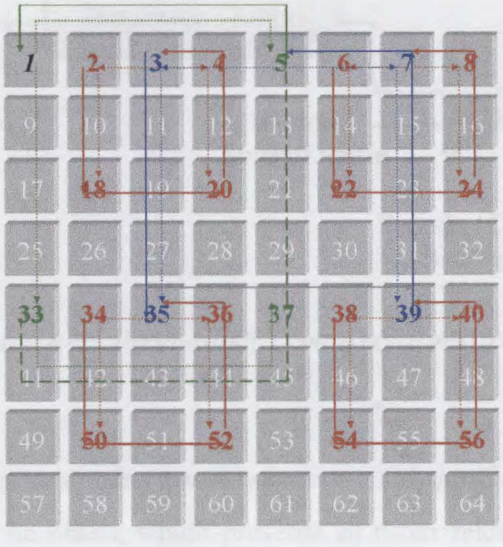


Figure 5-6: NBLOCK Signal Route Map

Figure 5-6 illustrates the interconnection pattern between pixels belonging to one of the three sub-trees found within an NBLOCK. If all three trees were mapped into this NBLOCK, it is easily observed that the more congested routing patterns exist within small areas (5 pixels) within the NBLOCK, which corresponds to the connections between high image frequency pixels.

5.2.3. Pixel-wise Tree Architecture

The pixel-wise tree architecture is presented in Figure 5-7. The signal *Sbout* is generated through simple combinational logic, while the *Pout* signal is based on the pixel operational mode (encode/decode) and the particular zerotree coding algorithm used (EZW/ZTE). The *Sbout* signal is set to logic 1 if *Cin*, *Sbin* or Pixel Significance (*Sig* Signal) is set to logic 1. The *Pout* signal, in encode mode, is generally dependant on the *Cin* signal, however, depending on the algorithm used *Pout* can also depend on the status of the Pixel Significance (*Sig*) . In decode mode the *Pout* signal is set to logic 1 only on the existence of special symbols that indicate child significance. The primary reason for this stems from the condition that, in decode mode, low frequency pixels dictate the significance of higher frequency pixels, resulting in a fully hierarchical decode approach.

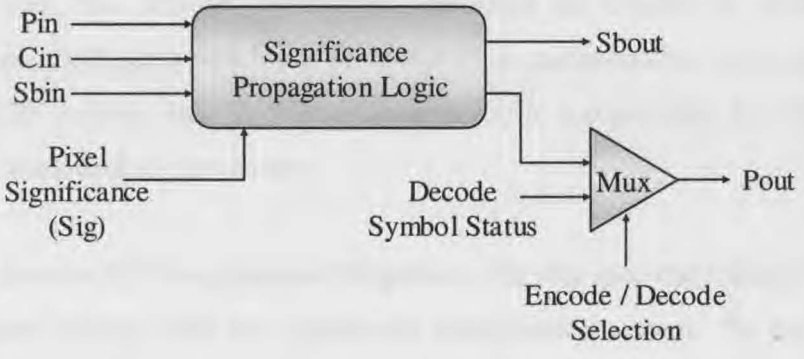


Figure 5-7: Pixel-wise Tree Architecture

The tree generation logic maintains a steady parent-child significance state that is based on the significance of the pixels, which presents all other relevant pixels with enough information to generate a zerotree based symbol when necessary.

5.3. *Embedded Zerotree Wavelet Codec*

The EZW coder, described in Chapter 3, is a two-phase coder that iteratively alternates between the symbol generation phase and coefficient refinement phase, gradually coding the wavelet coefficient to full precision during transmission. The hardware coder in turn is also designed to alternate between these two phases of generating symbols and successive approximation based refinement bits, both for transmission. Symbol generation is performed via a concurrent pixel-wise coefficient self-classification mechanism which spans the entire image-size array of pixels organised into multiple NBLOCKS. The parallel NBLOCK based significance propagation tree described in Section 5.2 provide all required significance information (from parent to children) for a particular pixel in an NBLOCK. The refinement bits, which result from successive approximation, are generated by bit-planing all the significant coefficients by means of right shifting coefficient registers. This process coupled with a significance identification mechanism, a symbol / refinement data extraction mechanism, two array interface buffers and a symbol based pixel activation mechanism, constitute the entire hardware architecture for the EZW codec. The VHDL code for a single pixel with th · EZW component is provided in Appendix A.

5.3.1. Coefficient Reorganisation

Once executed, the wavelet architecture described in Chapter 4, results in the generation of coefficients which are presented in an unfavourable orientation for use with the EZW architecture. At worst, the situation is compounded by two problems that require measures to compensate.

1. **Coefficient is 2's Complement Negative** – In this case the leading ones in the coefficient interfere with the significance identification system. To overcome this problem the negative coefficients are converted from 2's complement to sign and magnitude representation after the wavelet transform, but before progressing to the zerotree mode. Therefore, the shifting mode ($M1 = 0$ and $M2 = 0$) is used to perform this task. Given the amalgamation of circuits in Figure 4-27, Figure 4-28 and Figure 4-30, by establishing the control signals in Table 5-2 for a total of 9 clock cycles, this task can be performed. On the condition where the most significant bit R0_8 is logic 0 (i.e. a positive coefficient) the conversion circuitry is

disabled and a bypass is established. Also in order to maintain the sign in bit R0_8 for the zerotree architecture, R0_8 is not clocked. At the end of the 9 clock cycles any negative coefficient in R0 should be converted to a positive with the MSB set to the sign.

Table 5-2: Negative 2's Complement Convert Mode

Bold = Changed Signals		*Italics = Forced Signals*												
Mode Sel		Register R0										Adder	Clock	
M1	M2	Above	ClkMSB	EnR0	HPRin	LR	R0Ct1	R0Ct2	RC	REV	WT	S	Cycles	Comment
0	0	X	0	0	X	X	1	0	X	0	*0*	0	1	Set Carry
0	0	X	0	**1**	X	X	1	0	X	0	*0*	0	**8**	Convert

2. **Right Shift Presents LSB First** – Since R0 is configured to perform only right shifts and the coefficients are aligned with the LSB in bit position R0_0, it becomes necessary to reverse the bit-positioning of the coefficient in register R0. This is accomplished by performing the algorithm described in Section 4.7.2 within register R0. Table 5-3 lists the control configuration required to perform the bit-position reverse of R0 within 8 clock cycles. This phase is carried through in the pixel shift mode (*M1 = 0* and *M2 = 0*) for all pixels in the array.

Table 5-3: R0 Reverse Configuration

Bold = Changed Signals		*Italics = Forced Signals*												
Mode Sel		Register R0										Adder	Clock	
M1	M2	Above	ClkMSB	EnR0	HPRin	LR	R0Ct1	R0Ct2	RC	REV	WT	S	Cycles	Comment
0	0	X	0	1	X	X	0	0	X	1	*0*	X	1	Do Swap
0	0	X	0	1	X	X	0	0	X	**0**	*0*	X	**3**	Shift Half
0	0	X	0	1	X	X	0	0	X	**1**	*0*	X	1	Do Swap
0	0	X	0	1	X	X	0	0	X	**0**	*0*	X	**3**	Shift Half

Once EZW decoding has been performed these two steps are repeated in reverse order to re-establish the original wavelet coefficients.

5.3.2. Significance Identification Architecture

Once a pixel coefficient has been correctly oriented, the zerotree significance identification for the EZW encode phase can commence. As described in Chapter 3

the EZW algorithm uses a threshold level to determine if a coefficient is significant. In the hardware implementation this threshold value is fixed at $2^7 = 128$ and the coefficient, in each iteration, is constantly multiplied by two (left shifted) until it reaches the threshold level or until 8 shifts are made (i.e. the bit precision). As a result a coefficient is never permanently deemed insignificant, except in the case of a zero coefficient. For example if the value 100 (01100100b) is chosen as the coefficient, then in the first iteration, since the MSB is zero, this value is deemed insignificant. However, in the next iteration the value will become 200 (11001000b) and will be deemed significant.

The encode stage of the EZW significance identification architecture requires the use of a shifting register, such as R0, a memory cell to keep track of previous insignificant status and some control logic. Figure 5-8 is a block diagram illustrating the components associated with the EZW significance identification.

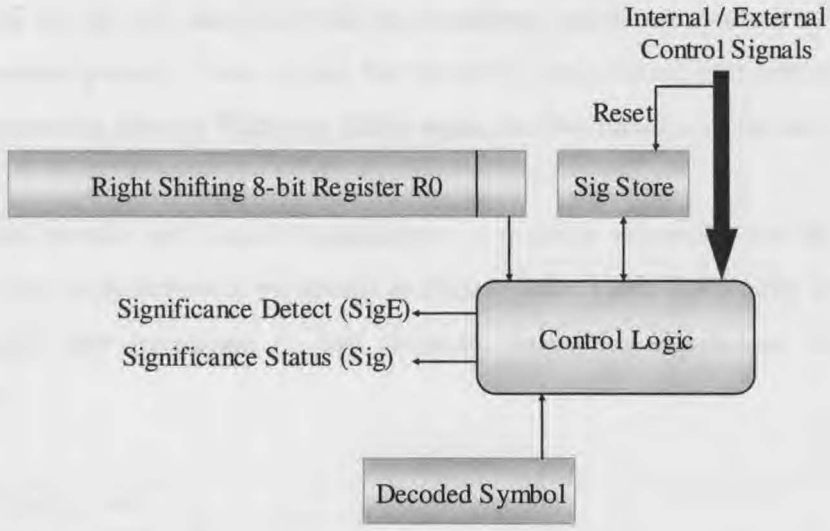


Figure 5-8: Significance Identification Architecture

The significance store, once reset in encode mode, is only set when the LSB (now carrying the MSB of the coefficient) of R0 receives a logic 1 after shifting. The moment the store is set, the coefficient in R0 is considered significant and this state is reflected through the Significance Detect signal *SigE* to all other components requiring this signal. In the following clock cycle this signal is switched logic low and the *Sig* (Significance Status) signal is latched to logic high. This indicates that the coefficient

has been found significant previously, in reference to the current iteration. This fulfils the requirement dictated by the EZW algorithm that once a coefficient has been found significant, in subsequent iterations, it is considered insignificant. This state is altered only when the reset signal is issued. In encode mode, The *SigE* signal is used to determine the symbols, while the *Sig* signal is used to determine the refinement bits required for EZW successive approximation. The *SigE* signal is also linked to the significance tree generation architecture to propagate the significance of the originating pixel to both its parent and children.

The EZW decode mode also relies on a similar architecture to identify significant pixels. This therefore, allows for the use of the same architecture, albeit, one that switches inputs depending on the operating mode. The decode mode also employs the two signals *SigE* and *Sig* for the same purpose as in the encode mode. However, the significance is determined on the reception of an appropriate significant symbol rather than the contents of register R0. From Chapter 3, it is known that the symbols, ***POS*** and ***NEG***, are the only two that could be considered significant symbols as they initiate the refinement-passes. Once a pixel has received one of these two symbols the pixel can never receive either a ***POS*** or a ***NEG*** again, for the duration of the decode cycle.

Given this encode and decode requirement, a possible schematic for the significant identification architecture is presented in Figure 5-9. Table 5-4 briefly describes the I/O signals that propagate to and originate from the significance identification schematic.

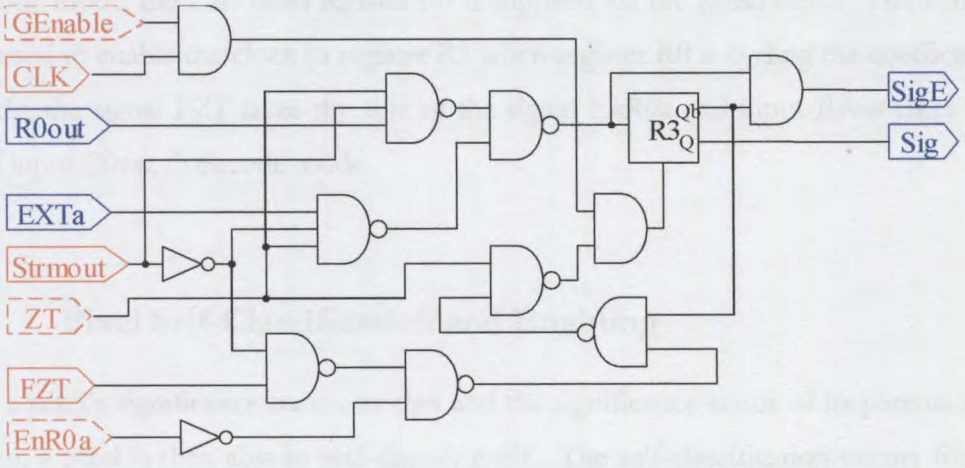


Figure 5-9: Significance Identification Schematics

In order for the circuit to function as intended, it is required that register R3 be initialised to logic 0. This is accomplished by issuing one or more clock cycles when the circuit is not in zerotree mode (i.e. $ZT = 0$). This has been designed with the knowledge that the pixel will be performing at least one clock cycle in the wavelet mode ($WT = 1$) or the Motion Compensation mode ($MC = 1$) or the Shift mode ($SH = 1$) before the zerotree mode commences. In these cases the ZT signal is set low.

Table 5-4: Significance Identification Circuit I/O

Signal	Dir	Type	Description
Clk	In	Control	Global Clock Signal
EnR0a	In	Control	Enable R3 Clock On R0 Shift
FZT	In	Control	Enable R3 Clock On Zerotree Formation
GEnable	In	Control	Global Enable
Strmout	In	Control	Encode / Decode Select (1 = Encode)
ZT	In	Control	Zerotree Mode Select
R0out	In	Data	Bits Originating From Register R0
EXTa	In	Data	Significant Symbol In
SigE	Out	Status	Initial Significance Detected

Once register R3 has been initialised to 0, zerotree encoding or decoding can commence by setting signal $ZT = 1$ (This is controlled by the mode selection circuity described in Section 4.7.4.) The *Strmout* signal, also used in the motion compensation unit, is reused to dictate if the circuit is in encode (*Strmout* = 1) or decode (*Strmout* = 0) mode.

In encode mode, the LSB from register R0 is supplied via the *R0out* input. The *EnR0a* signal used to enable the clock to register R3 when register R0 is cycling the coefficient. Similarly, the signal *FZT* takes the role of the signal *EnR0a* and input *R4out* takes the role of input *R0out*, in decode mode.

5.3.3. Pixel Self-Classification and Enabling

Given a pixel's significance status, its sign and the significance status of its parents and children, a pixel is then able to self-classify itself. The self-classification occurs for all pixels in the entire array in parallel. A pixel can be classified into one of four states or

be completely disabled. Table 5-5 lists all these possible states. Since there are four states, two bits (signals *ZS1* and *ZS2*) are used to represent the symbols.

Table 5-5: EZW Self-Classification Symbols

Pin	Sign	Cin	SigE	Description	Symbol	ZS1ZS2
1	0	0	0	Zerotree Root	ZTR	00
1	0	0	1	Positive Significance	POS	10
1	0	1	0	Isolated Zero	IZO	01
1	0	1	1	Positive Significance	POS	10
1	1	0	0	Zerotree Root	ZTR	00
1	1	0	1	Negative Significance	NEG	11
1	1	1	0	Isolated Zero	IZO	01
1	1	1	1	Negative Significance	NEG	11
0	X	X	X	Pixel Disabled	ZTR	00

A ***ZTR*** symbol is generated by a pixel which has a significant parent or sibling, is not currently significant and contains no descendant pixels that are significant. This symbol results in the deactivation of the *Pout* (Signal propagating to all descendants confirming the significance of the parent) signal to logic 0.

A ***POS*** symbol is generated by a pixel which contains a significant positive coefficient during the current iteration. The significance statuses of a pixel's descendants have no bearing on the generation of this symbol. However, this symbol implies that a pixel's immediate descendants should generate their own symbols; therefore, it results in the activation of the *Pout* signal to logic 1.

A ***NEG*** symbol is generated by a pixel which contains a significant negative coefficient during the current iteration. The significance statuses of a pixel's descendants have no bearing on the generation of this symbol. However, this symbol implies that a pixel's immediate descendants should generate their own symbols; therefore, it results in the activation of the *Pout* signal to logic 1.

An ***IZO*** symbol is generated by a pixel which is insignificant in the current iteration but contains at least one significant descendant. This symbol results in the activation of the *Pout* signal to logic 1.

Given these conditions, Figure 5-10 presents a possible schematic for the generation of the two symbol bits. The additional signal, *Strmout*, is employed to clear *ZS1* and *ZS2* to logic 0, which is a requirement during decode (i.e. *Strmout* = 0) mode, as no symbols are generated in decode mode, only received.

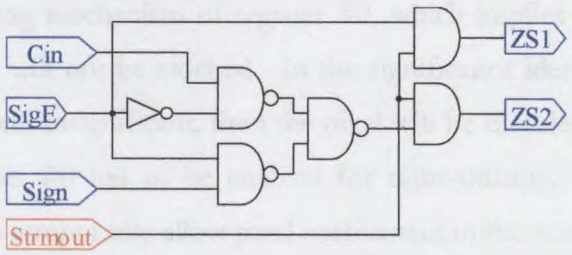


Figure 5-10: EZW Classification Schematics

Although the schematic is capable of generating all four classification symbols it is not capable of providing a no transmission state. When the status signal *Pin* is low, this indicates that the parent is a zerotree and that all descendants belonging to that parent should be excluded from being able to transmit a symbol. This is overcome by having a global pixel enable/disable mechanism (a new *GEnable* signal) that has ties with the scale control architecture described in Section 4.6.2. In accordance with the scale control architecture, a pixel is enabled for processing when both the *VEnable* and *HEnable* signals arriving at a pixel is set high. This scheme is now modified to include pixel enablement only when all three signals *VEnable*, *HEnable* and a new signal named *ZTEnable*, are all set high. The *ZTEnable* signal only influences the pixel enablement during the zerotree phase. The resultant *GEnable* signal, which is derived from these three signals, propagates to all components requiring this global enable signal.

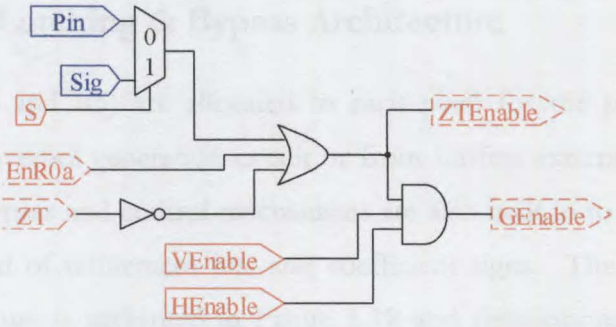


Figure 5-11: EZW Pixel Enable Schematics

Figure 5-11 shows the pixel enable schematics employed within each pixel. It is clear to see that when the pixel is not in zerotree mode ($ZT = 0$), that $ZTEnable = 1$, which excludes the zerotree circuitry from interfering with other modes. In zerotree mode, the *ZTEnable* signal is directly controlled by a further two signals, the *EnR0a* signal and the output from the multiplexer. Figure 4-28 shows that the generated *GEnable* signal influences the clocking mechanism of register R0, which implies that when a pixel is disabled, register R0 will not be clocked. In the significance identification phase, if a pixel is currently found insignificant, then the pixel will be disabled. However, for the next iteration, register R0 has to be enabled for right-shifting, therefore, the signal *EnR0a* is arranged to temporarily allow pixel enablement in the zerotree mode.

The *Pin* and *Sig* signals also influence the status of the *ZTEnable* signal via the multiplexer. The *S* signal, originally used for summation select in the adder (Section 4.7.1) is reused to select between the symbol generation mode and the successive approximation refinement bit generation mode, as the adder is not used concurrently with the zerotree mode. In both encode and decode modes, if a parent pixel is significant, then the EZW algorithm dictates that that parent's immediate descendants are required to generate or receive a symbol. Therefore, the *Pin* signal is employed to control pixel activation when in symbol generation/receive mode. A pixel generates or receives a refinement bit only when it has been identified as significant previously and therefore, the *Sig* signal is used to control the pixel enablement in refinement mode.

The enablement of a pixel in any zerotree mode implies that it contains or awaits a valid symbol or refinement bit and is ready for transmission or reception.

5.3.4. Pixel Latching & Bypass Architecture

Two registers (R4 and R5) are allocated to each pixel for the purpose of symbol capture from the symbol generation circuit or from buffers external to the array. In addition to this, bypass and control mechanisms are also built in to aid in the zerotree extraction and load of refinement bits and coefficient signs. The schematic for the proposed architecture is presented in Figure 5-12 and descriptions of the associated I/O ports are listed in Table 5-6. For any data to be loaded into these registers, the

signals *GEnable* and *EnR0a* are required to contain the values of logic 1 and 0 respectively. Loading is performed by clocking the circuit in this state.

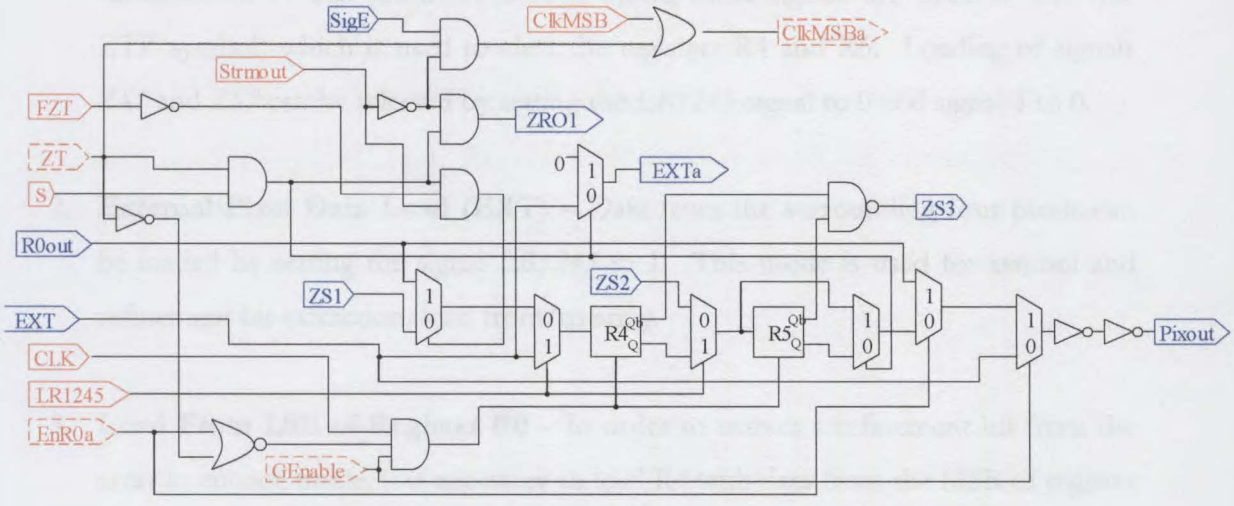


Figure 5-12: Symbol Latch and Bypass Schematic

Table 5-6: I/O Ports for Symbol Latch

Signal	Dir	Type	Orig/Dest	Description
CLK	In	Clock	External	Global Clock Signal
ClkMSB	In	Control	External	Force Enable R0 MSB Clocking
FZT	In	Control	External	Form Zerotrees
LR1245	In	Control	External	Select Load for R4 & R5
S	In	Control	External	Symbol / Refinement Select
Strmout	In	Control	External	Encode / Decode Select
ClkMSBa	Out	Control	Internal	Enable R0 MSB Clocking
EnR0a	In	Control	Internal	Register R0 Clock Enable / ZT Bypass
GEnable	In	Control	Internal	Pixel-wise Enable
ZT	In	Control	Internal	Zerotree Mode Select
EXT	In	Data	External	Data From Surrounding Pixels
EXTa	Out	Data	Internal	Data For Decode Significance Detect
Pixout	Out	Data	External	Data To Surrounding Pixels
R0out	In	Data	Internal	Data From LSB of Register R0
SigE	In	Data/Cont	Internal	Significance Detect Signal
ZR01	Out	Control	Internal	Place Refinement Into R0
ZS1	In	Data	Internal	Symbol bit 1
ZS2	In	Data	Internal	Symbol bit 2
ZS3	Out	Data/Cont	Internal	Parent Significance During Decode

Registers R4 and R5 receive data from one of three sources.

1. **Signals *ZS1* and *ZS2*** – These signals originate from the symbol classification circuitry (Section 5.3.3) and, in encode mode, contain a symbol pertaining to the classification of that pixel. In decode mode, these signals are fixed at zero (i.e. ZTR symbol) which is used to clear the registers R4 and R5. Loading of signals *ZS1* and *ZS2* can be selected by setting the *LR1245* signal to 0 and signal *S* to 0.

2. **External Pixel Data Load (EXT)** – Data from the surrounding four pixels can be loaded by setting the signal *LR1245* to 1. This mode is used for symbol and refinement bit extraction/load from/to array.

3. **Load From LSB of Register R0** – In order to extract a refinement bit from the array in encode mode, it is necessary to load R4 with data from the MSB of register R0. To accomplish this, the four signals *ZT*, *FZT*, *Strmout*, *LR1245* and *S* require to be set to logic 1, 0, 1, 0 and 1 respectively. The *FZT* signal is used to engage/disengage this mode.

In addition to this function the architecture has three bypass modes

1. **Pixel Bypass Mode** – When a pixel is disabled, with *GEnable* set low, the pixel acts as a link joining adjacent pixels on both sides together (i.e. *EXT* is connected to *Pixout*). In this mode no data reaches the pixel or is generated by the pixel. The two series inverters allow for signal boost on long distance bypasses if conventional transmission gate based multiplexers are used.

2. **Register R4 & R5 Bypass** – For wavelet functionality, when register R0 is activated (signal *EnR0a* is set high) registers R4 and R5 are disconnected to connect the LSB of register R0 instead to *Pixout*. Also when *EnR0a* is set high clocking to registers R4 and R5 are also blocked.

3. **Bypass Register R5** – When extracting and loading refinement bits, which are one bit in length, it becomes necessary to utilise one-bit registers. For this reason register R5 can be bypassed by setting the *S* signal to 1. In addition, if the circuit is

in decode mode (*Strmout* = 0), then the *ZRO1* signal is employed to feed refinement bits to register R0.

Finally, the proposed architecture performs two other functions.

1. **Enable MSB Clock of R0 When Significant** – In addition to being able to force the signal *ClkMSBa* high via *ClkMSB*, in decode mode (*Strmout* = 0), the architecture allows for the enablement of Register R0's MSB clock when the signal *SigE* becomes high. This operation is used to load the sign into R0_8 when a significant symbol is decoded.

2. **Parent Decode Symbol** – When in decode mode, a parent pixel, via the *Pout* signal, informs its descendants to prepare for symbols when it receives a significant symbol (***POS***, ***NEG*** or ***IZO***). The *Pout* signal is derived from the *ZS3* signal. The binary representations of ***POS***, ***NEG***, ***IZO*** and ***ZTR*** are 10, 11, 01 and 00 respectively. The *ZS3* signal should only be low when the symbol is a ***ZTR*** (00b) which is the same as saying the *ZS3* signal should be low, only when the inverse of both registers R4 and R5 result in 11b, hence the usage of a NAND gate.

5.3.5. Significance Tree Signal Generation

The per-pixel architecture required to populate the significance tree is chiefly governed by the zerotree algorithm. There are two outputs, namely *Sbout* and *Pout* signals, which are associated with the generation of the EZW significance tree. The *Sbout* signal or sibling out signal is derived from the three signals *Cin*, *Sbin* and *SigE* and is switched high if any of these signals are high. Both *Cin* and *SigE* when high generate significant symbols (***POS***, ***NEG*** or ***IZO***), and as such require transmitting. This is conveyed via *Sbout* to the parent. The *Sbin* signal simply acts as a linking signal that provides the collective significance of all siblings prior to the current.

The *Pout* signal, however, is dependant on which operational mode the pixel is in. In encode mode the *Pout* signal is derived high from the existence of a high status between the two signals *Cin* and *SigE*. The *Cin* signal presents significance information

about the children to the parent. If a child is significant, then the parent is required to enable all children pixels and prepare them for symbol transmission. Similarly, since the symbols ***POS*** and ***NEG***, derived from the high status of the *SigE* signal, do not convey the significance information of the descendants, the parent is required to force the children to generate a symbol. In decode mode, the *Pout* signal is set high when the parent's received symbol corresponds to one of ***POS***, ***NEG*** or ***IZO***. Therefore, when the signal *Strmout* is set low (for decode) the *Pout* signal assumes the content of *ZS3* which is derived from the symbols according to Section 5.3.4.

The proposed architecture to perform this function is presented in Figure 5-13

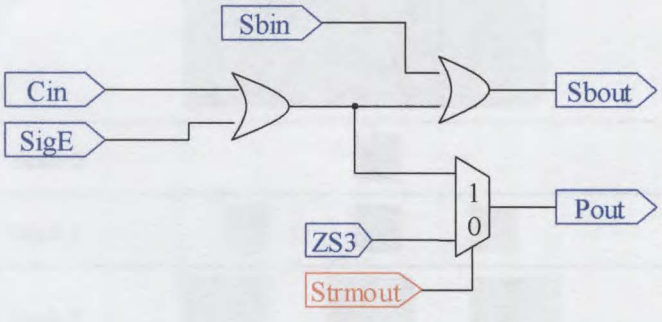


Figure 5-13: Per-pixel Significance Tree Generation

5.3.6. Data Extraction/Load Architecture

Two phases of operation that require some architectural consideration are the extraction and load of compressed zerotree data from/to the array, and ultimately each pixel. Data extraction is performed when the pixel contains symbols or refinement bits to be transmitted, while the load is performed to populate the array with the same data prior to decoding.

To be able to correctly decode encoded streams, both encode and decode processes are conducted in a subband based hierarchical manner, which also provides a mechanism for scalable video resolution selection. A 3-scale image decomposition system consists of 10 subbands to be transmitted as presented in Figure 5-14. The transmission/reception subband order follows that of LL, LH_1, HL_1, HH_1, LH_2, HL_2, HH_2, LH_3, HL_3 and HH_3. Each of these subbands are selected by applying the

appropriate control signals to the *VEnable* and *HEnable* signal lines as described in section 4.6.2. Once a particular subband has been enabled, the pixel data for that subband is raster scanned and extracted at the bottom of the array via a series of downward shifts. During decode, the raster scanning mechanism is activated at the top edge of the array yet the data is shifted in a downward direction into the array.

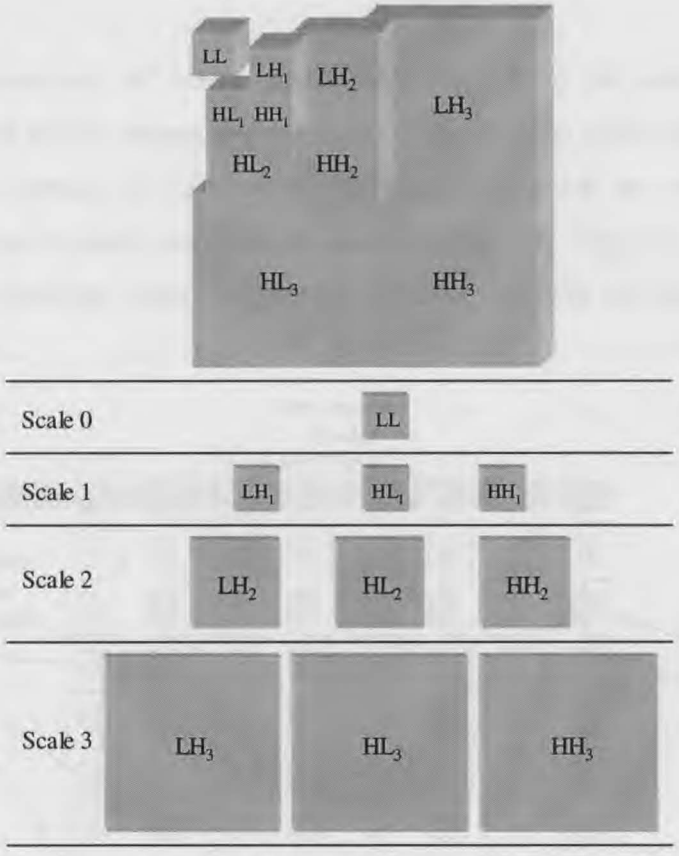


Figure 5-14: Subband Encode Decode Order

In encode mode, pixels resulting from a disabling scale control sequence or coefficient insignificance switch into bypass mode, therefore the array is effectively reduced to only pixels that require transmission. Extraction of data from the array in this state is the most efficient, in terms of clock cycles. However, it introduces a significant problem in pixel position synchronisation as the number of enabled pixels present per column is unknown to the control mechanism. In decode mode, a count of active pixels must be performed to enable correct decoding. One technique that may be used is to introduce a series of counters at the top and bottom of the array. However, these present a significant increase in complexity in the control architecture and the array-

edge components. The proposed alternative is to provide a distributed counting mechanism within each pixel by allowing each pixel to add a link to a chain of bits that can be shifted in parallel to the data. To realise this architecture each pixel is outfitted with an additional register and some control logic. The control logic allows for the register orientation to be switched between its function as a component of a large column-wise shift-register or as a pixel-wise single cell storage element.

Using this architecture, all active pixels place logic 1 in the storage cell, while deactivated pixels simply bypass the cell completely. The net effect that occurs is the generation of a number of column-wise shift-registers, which are composed of the storage cells of active pixels only, and all containing logic 1. Figure 5-15 illustrates an example of this counting method for an 8 by 8 NBLOCK with the HH_3 band selected for transmission.

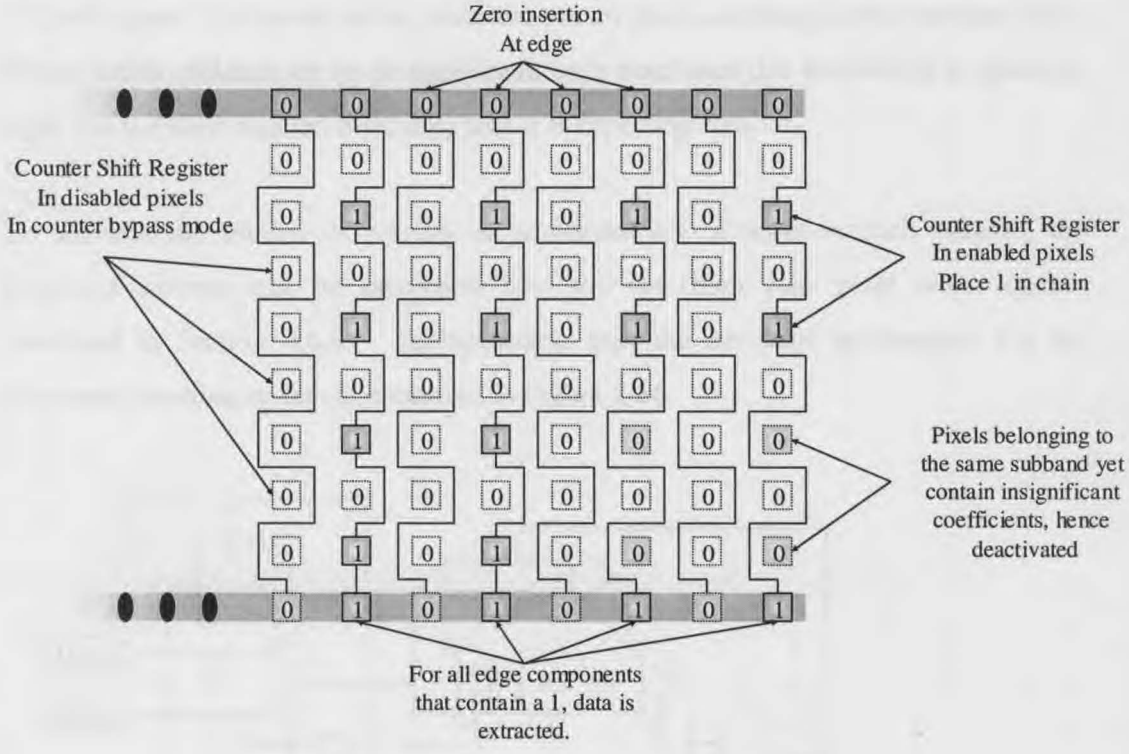


Figure 5-15: Distributed Pixel Counting

Although the entire HH_3 subband is selected, four pixels within this subband are deactivated, as would happen if the pixels were not carrying any significant coefficients. At the top edge of the array, logic 0's are fed into the shift register, the only 0's to traverse the system. When these 0's propagate to the bottom-edge of the array via the

large shift register, this indicates that the column has exhausted all of its significant coefficients and, as such, expectation of any data from that column should be halted. In this light, out of the four active columns in Figure 5-15, the two left-hand columns will contain data for four clock iterations, while the two right-hand columns will contain data for only two clock iterations. This ensures the extraction of exactly 12 sets of data. Using this self-counting system any non-symmetrically positioned data array is efficiently extracted.

The same system is employed in the decoding process, with the exception of the shift register, which now propagates upwards instead of downwards. Also the end of column zeros are inserted at the bottom edge instead of at the top. When a zero propagates to the top, via the shift register, it indicates that the column has no more free pixels requiring coefficients, and the entire column is disabled via that column's *VEnable* signal. In decode mode, since the parent pixels, residing in the previous scale, dictate which children are to be significant, each pixel uses this knowledge to generate logic 1 in the shift register, indicating that it is expecting data.

To alleviate the burden of routing an additional set of wires in each column, the proposed scheme can be integrated into the low/high pass pixel select system described in Section 4.6.3. Incorporating this, the per-pixel architecture for the proposed counting system is presented in Figure 5-16.

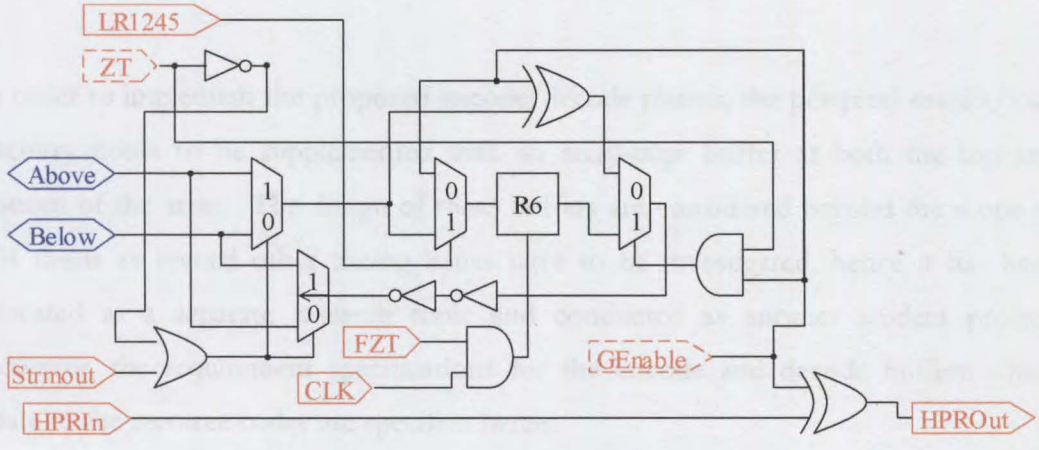


Figure 5-16: Per-pixel Counting Architecture

Initially, when the circuit is not in zerotree mode (ZT = 0), it is disabled, and acts the part of two XOR gates, which invert the *Above* and *HPRIn* signals when *GEnable* (i.e.

Pixel is active) is held high. This complies with the wavelet transform which requires a low/high pass pixel selection mechanism as described in Section 4.6.3. When in zerotree mode the functions of the *Above* and *Below* signals change to become pixel interconnections. The functions of *HPRIn* and *HPROut* do not change. To support both encode and decode, the signals *Above* and *Below* are required to be bidirectional. In encode mode the shift-register is configured to accept data from *Above* and release data to *Below*. In decode mode the opposite is true.

Investigating the encode phase, when loading the symbols or refinement bits into register R4 or R5, the *GEnable* (determined by the significance tree) signal is also loaded into register R6 by setting the signals *LR1245* and *FZT* to 0 and 1 respectively. Once this status is in register R6, setting signals *LR1245*, LR and RC (from Section 4.5) to 1, 0 and 1 respectively will result in interconnection of two array-wise shift-registers, one for the symbol/refinement bits and the other for the counting circuitry. If extracting symbols, for every two-bit symbol extracted, the counting is shifted once, while if extracting refinement bits, for each bit extracted the counter is again shifted once.

The operational principal for the decode phase is identical, except for an exchange between the functions of *Above* and *Below*.

5.3.7. Array Edge Buffer Design

In order to implement the proposed encode/decode phases, the per-pixel extract/load circuitry needs to be supplemented with an array-edge buffer at both the top and bottom of the array. The design of these buffers are considered beyond the scope of this thesis as several other timing issues have to be investigated, hence it has been allocated as a separate research topic and conducted as another student project. However, the requirement specifications for the encode and decode buffers which relate to the zerotree codec are specified herein.

The proposed placement of the encode buffer is at the bottom of the array and it requires cells equalling the number of pixels in the bottom row of the array. Each cell is connected individually to a corresponding pixel by two connections. The first, a

unidirectional connection linking the pixel data output to the buffer cell data input, and the second, a bidirectional link connecting the *Bottom* signal of a last row pixel and its corresponding signal (named here as *Statin*) in the buffer cell. In symbol extraction mode (Encode), all buffer cells should be ready to accept a two-bit symbol from the pixel above it and a single bit from the *Statin* signal, when the array has finished generating symbols. Once this data is received it can be serially removed from the buffer, at a higher clock speed, and transmitted. Only symbols that are accompanied by logic 1 in the *Statin* link when the first symbol bit arrives in the buffer are removed. All cells containing a zero are placed in bypass mode. The proposed encode buffer (Bottom) architecture is presented in Figure 5-17 and an I/O port description is provided in Table 5-7.

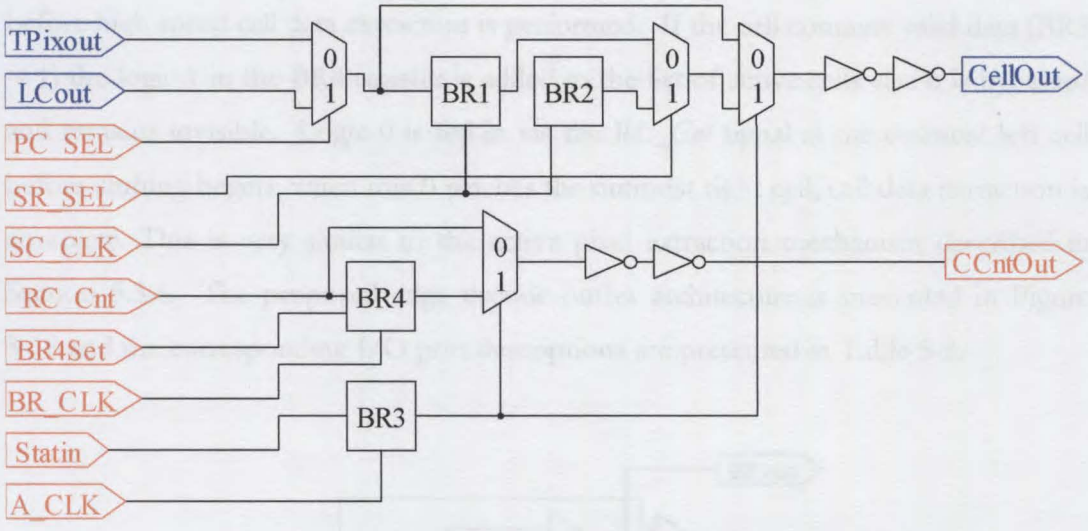


Figure 5-17: Edge Encode Buffer Architecture

The registers BR1 and BR2 are alternatively used to shift symbol or refinement bits depending on the status of *SR_SEL* (*SR_SEL* = 1, sets symbol mode). These registers should be able to be clocked at the array frequency (100 KHz) or a much higher buffer extract frequency (~ 1-5 MHz) depending wether data is being shifted from the array or from the adjacent cell respectively. The register B3 is used to capture the *Statin* signal and provide a bypass mechanism for the cell extract counter and data stream if *Statin* is logic 0, when extracting data from the array. This register is clocked at the array's operating frequency.

Table 5-7: Edge Encode Buffer I/O Ports

Signal	Type	Description
A_CLK	Clock	Array Clock
BR4Set	Control	Set Register BR4 To Logic 1
BR_CLK	Clock	High Speed Bit Removal Clock
PC_SEL	Control	Cell/Pixel Data Input Select
RC_Cnt	Control	Remove Cell Counter In
SC_CLK	Clock	Symbol/Cell Switchable Clock
SR_SEL	Control	Symbol/Refinement Mode Select
Statin	Control	Array Column Count In
CCntOut	Control	Cell Counter Output
LCout	Data	Left Cell Data Output
TPixout	Data	Data From Pixel Above
CellOut	Data	Data To Next Cell On Right

The B4 register forms part of an active cell counter. This register is set to logic 1 before high speed cell data extraction is performed. If the cell contains valid data (BR3 = 1) the logic 1 in the BR4 register is added to the list of active cells else it is bypassed and remains invisible. Logic 0 is fed in via the *RC_Cnt* signal at the outmost left cell before shifting begins, when this 0 reaches the outmost right cell, cell data extraction is complete. This is very similar to the active pixel extraction mechanism described in Section 5.3.6. The proposed edge decode buffer architecture is presented in Figure 5-18 and the corresponding I/O port descriptions are presented in Table 5-8.

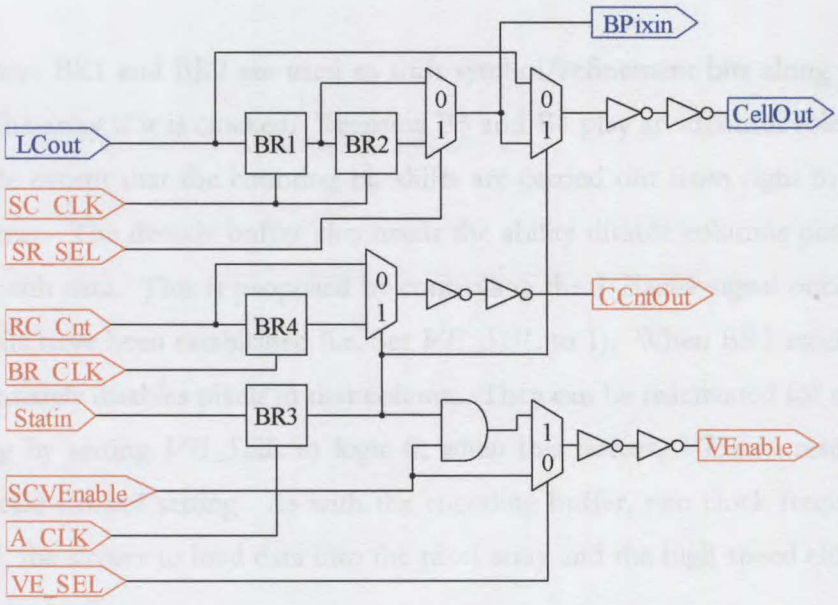


Figure 5-18: Edge Decode Buffer Architecture

Table 5-8: Edge Decode Buffer I/O Ports

Signal	Type	Description
A_CLK	Clock	Array Clock
BR_CLK	Clock	High Speed Bit Removal Clock
RC_Cnt	Control	Remove Cell Counter In
SC_CLK	Clock	Symbol/Cell Switchable Clock
SCVEnable	Control	VEnable Signal from Scale Control
SR_SEL	Control	Symbol/Refinement Mode Select
Statin	Control	Array Column Count In
VE_SEL	Control	Vertical Enable Select
VEnable	Data	Column Enable
CCntOut	Control	Cell Counter Output
LCout	Data	Left Cell Data Output
Bpixin	Data	Data To Pixel Below
CellOut	Data	Data To Next Cell On Right

The edge decode buffer is similar in nature to the encode buffer. However, the *Statin* signal belonging to a buffer cell is connected to the *Above* signal on a pixel at the top of the array and the *BPixin* signal connects to the *From_Top* input of that same pixel. The cell array is fed data from the left side of the buffer; however, the counter bit-shift propagation occurs from right to left. This is to provide the maintenance of data in correct order and supply a count to the receive decoder. Figure 5-19 illustrates the interfacing between the array and the two buffers.

The registers BR1 and BR2 are used to shift symbol/refinement bits along the buffer and into the array if it is clocked. Registers B3 and B4 play an identical role as that in the encode except that the counting bit shifts are carried out from right to left along the cell array. The decode buffer also needs the ability disable columns pixels as they are filled with data. This is proposed by controlling the *VEnable* signal once the array counter bits have been established (i.e. Set *VE_SEL* to 1). When BR3 receives a zero this immediately disables pixels in that column. They can be reactivated for scale based processing by setting *VE_SEL* to logic 0; when this occurs, *VEnable* resembles the original scale control setting. As with the encoding buffer, two clock frequencies are employed, the slower to load data into the pixel array and the high speed clock to load data into the buffer.

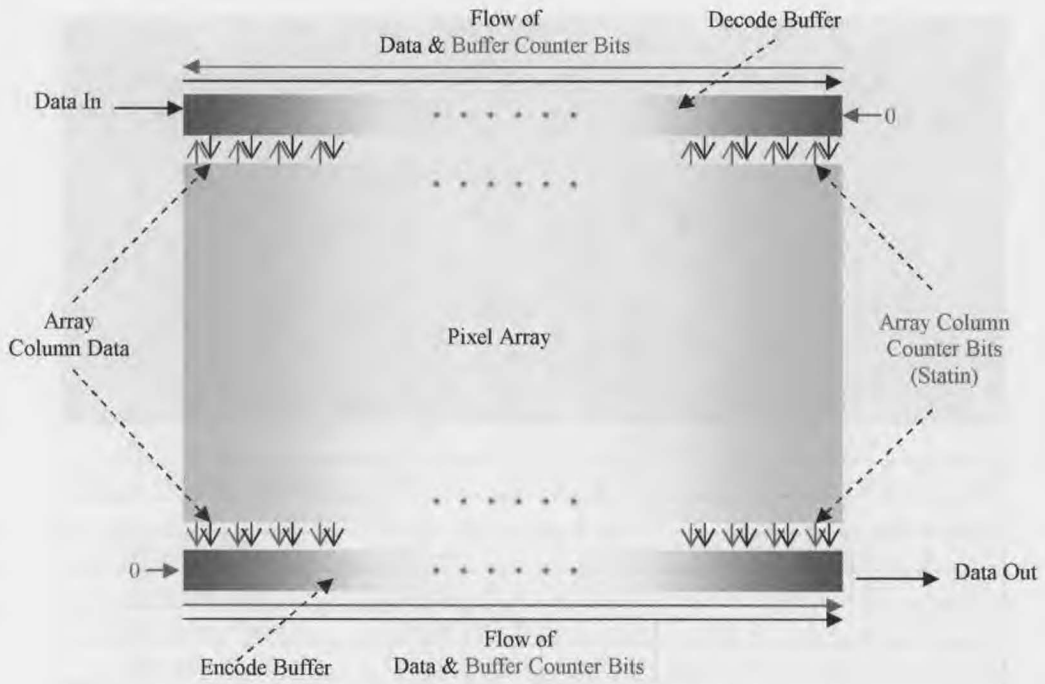


Figure 5-19: Buffer - Array Interfacing

5.3.8. The Encode Control Sequence

Control settings for the EZW full array encode cycle are presented in Table 5-9. The scale control steps (subband selection) are derived according to Section 4.6.2.

5.3.9. The Decode Control Sequence

The complete array-wise decode cycle is performed by following the control settings in Table 5-10. The scale control steps (subband selection) are derived according to Section 4.6.2.

Table 5-9: Encode Cycle

Phase	Description	M1	M2	R0Ct1	R0Ct2	LR	RC	S	EnR0	Strmout	LR1245	Rev	Fzt	ClkMSB			ClockCycles
1	Set Carry	0	0	1	0	0	0	0	0	0	0	0	0	0			1
2	2's Comp Convert	0	0	1	0	0	0	0	1	0	0	0	0	0			8
3	Reverse Phase 1	0	0	0	0	0	0	0	1	0	0	1	0	0			1
4	Reverse Shift	0	0	0	0	0	0	0	1	0	0	0	0	0			3
5	Reverse Phase 2	0	0	0	0	0	0	0	1	0	0	1	0	0			1
6	Reverse Shift	0	0	0	0	0	0	0	1	0	0	0	0	0			3
7	Latch Symbols	1	1	0	0	0	0	0	0	1	0	0	0	0			1
8	Load R6	1	1	0	0	0	0	0	0	1	0	0	1	0			1
9	**Select Lowest Frequency Subband**																
10	Shift Symbol LSB	1	1	0	0	0	1	0	0	1	1	0	0	0			1
11	Shift LSB & Counter	1	1	0	0	0	1	0	0	1	1	0	1	0			1
12	**Repeat Phases 10 & 11 until last pixel in all columns have a 0 for Below**																
13	**Repeat Phases 10, 11 & 12 for each subband**																
14	**Select Entire Array**																
15	Set R6 & Ref bit	1	1	0	0	0	1	1	0	1	0	0	1	0			1
16	**Select Lowest Frequency Subband**																
17	Shift LSB & Counter	1	1	0	0	0	1	1	0	1	1	0	1	0			1
18	**Repeat Phase 15 until last pixel in all columns have a 0 for Below**																
19	**Repeat Phases 16 & 17 for each subband**																
20	**Select Entire Array**																
21	Shift R0 for Next Bit	1	1	0	0	0	1	0	1	1	0	0	0	0			1
22	**Repeat 7 to 21 until mode change or R0 has done 8 shifts**																

Table 5-10: Decode Cycle

Phase	Description	M1	M2	R0Ct1	R0Ct2	LR	RC	S	EnR0	Strmout	LR1245	Rev	Fzt	ClkMSB			ClockCycles
1	Clear R3	0	0	0	0	0	0	0	0	0	0	0	0	0			1
2	Clear R4 & R5	1	1	0	0	0	1	0	0	0	0	0	0	0			1
3	**Enable Lowest Frequency Subband**																
4	Load R6	1	1	0	0	0	1	0	0	0	0	0	1	0			1
5	Load Symbol LSB	1	1	0	0	0	1	0	0	0	1	0	0	0			1
6	Load Symb MSB & R6	1	1	0	0	0	1	0	0	0	1	0	1	0			1
7	**Repeat Phases 5 & 6 until 0 is received at the top pixel via Above**																
8	**Repeat Phase 4 to 7 for each subband**																
9	**Enable All Pixels**																
10	Load R0 Sign	1	1	0	0	0	1	0	1	0	0	0	0	1			1
11	**Enable Lowest Frequency Subband**																
12	Load R6 for Ref Bit	1	1	0	0	0	1	1	0	0	0	0	1	0			1
13	Load Ref Bit	1	1	0	0	0	1	1	0	0	1	0	1	0			1
14	**Repeat Phase 13 until 0 is received at the top pixel via Above**																
15	**Repeat Phase 13 to 14 for each subband**																
16	**Enable All Pixels**																
17	Load R0 Ref Bit	1	1	0	0	0	1	1	1	0	0	0	0	0			1
18	**Repeat 3 to 17 until mode change or R0 has done 8 shifts**																
19	Reverse Phase 1	0	0	0	0	0	0	0	1	0	0	1	0	0			1
20	Reverse Shift	0	0	0	0	0	0	0	1	0	0	0	0	0			3
21	Reverse Phase 2	0	0	0	0	0	0	0	1	0	0	1	0	0			1
22	Reverse Shift	0	0	0	0	0	0	0	1	0	0	0	0	0			3
23	Set Carry	0	0	1	0	0	0	0	0	0	0	0	0	0			1
24	2's Comp Convert	0	0	1	0	0	0	0	1	0	0	0	0	0			8

5.4. Zerotree Entropy Coder

The ZTE coder, unlike the EZW, is a two-pass coder that performs a single symbols and coefficients pass before transmitting the contents of the array. The hardware coder, therefore, performs only two passes per encode of the entire array. As with the EZW coder, symbols for all pixels in the array are generated via a concurrent pixel-wise coefficient self-classification architecture. Highly parallel significance tree propagation techniques are employed to gather significance data from all pixels within a single NBLOCK (as described in Section 5.2) to enable the generation of symbols. The symbols are supplemented with a set of coefficients which, unlike the EZW, does not require special preparation. The coefficients, however, do require a quantisation phase, because the ZTE does not support successive approximation. This process coupled with a significance identification mechanism, a symbol / coefficient data extraction mechanism, two array interface buffers and a symbol based pixel activation mechanism, constitute the entire hardware architecture for the ZTE codec. The VHDL code for a single pixel with the ZTE component is provided in Appendix B.

5.4.1. Coefficient Quantisation

As mentioned previously (Section 3.3.1), the ZTE algorithm requires the use of a subband quantisation component. The ZTE algorithm relies quite heavily on the generation of grouped zero (or -1) coefficients. The hardware implementation of the subband quantisation scheme requires the ability to right-shift the coefficient (register R0), hold the coefficient MSB and select individual subbands. The architecture proposed in Chapter 4 exceeds the specifications required to perform this function.

To perform quantisation, firstly a subband is selected, then that subband is right-shifted appropriately to match the required quantisation. Since right-shifts are used the possible quantisation coefficients are factors of two. Table 5-11 lists the required control sequence for performing quantisation once a subband has been selected. The value X represents the number of shifts required, for instance if the subband is to be quantised by a factor of 4 then $X = \log_2 4 = 2$.

Table 5-11: Quantisation Select

Mode Sel		Register R0								Clock	
M1	M2	R0Ct1	R0Ct2	EnR0	Strmout	LR1245	Rev	FZT	ClkMSB	Cycles	Comment
1	0	0	0	1	1	0	0	0	0	X	X = Number of Right-shifts

Negative coefficients which are truncated in this fashion never reach zero instead they reach the value of -1. This is because two's complement truncation results in the generation of a long string of ones which represent the value -1. This condition needs to be accounted for.

5.4.2. Significance Identification Architecture

Unlike the complex scheme used by the EZW algorithm, the ZTE algorithm only requires a simple significance identification architecture. This is because it only requires knowledge of zero (or -1) coefficients. To account for the case of negative numbers being truncated to negative one, any truncated negative one coefficients are considered insignificant also. Therefore, the search for insignificant coefficients requires the evaluation of both conditions; the coefficient is zero or the coefficient is negative one. Negative one in 8-bit 2's complement binary is represented as 11111111b, while zero is represented by 00000000b. Therefore, the condition becomes a case of determining if all bits of the coefficient are the same or not. This is easily accomplished by an XOR logic gate. If the XOR triggers as the contents of the coefficient are cycled through the XOR gate, then the coefficient can be deemed significant, as it is neither zero nor negative one. The value resulting from the XOR can be stored as the significance status. Figure 5-20 illustrates the proposed architecture. The register R0 described in Section 4.7.2 is slightly modified to provide the two extra outputs *R0_0out* and *R0_1out*. The latch is reset only when the pixel exits the zerotree processing mode (ZT = 0). Therefore, the significance status is maintained for the duration of the zerotree process.

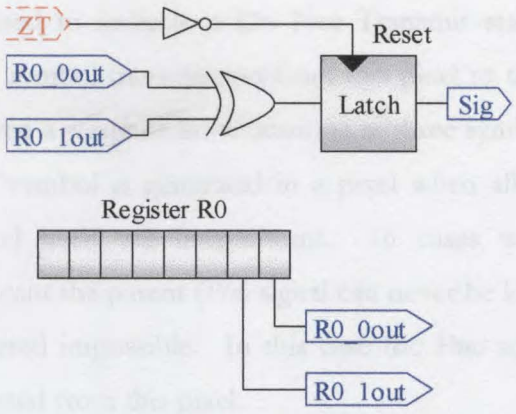


Figure 5-20: ZTE Significance Identification Architecture

5.4.3. Pixel Self-Classification Architecture

Once each pixel has completed identifying self-significance identification, this information is propagated to all relevant pixels via the significance tree. When each pixel receives this significance data, which occurs concurrently throughout the entire array within one clock cycle, the pixel is able to self-classify itself. The ZTE algorithm employs three symbols (***ZTR***, ***VZT*** and ***VAL***) for the generation of the significance tree and it is generated only once for the entire compression cycle of the image. Table 5-12 lists these symbols, the generation conditions and two-bit (*ZS1*, *ZS2*) representations.

Table 5-12: ZTE Self-Classification Symbols

Pin	Cin	Sig	Description	Symbol	ZS1ZS2
0	0	0	Do Not Trasmit	DNT	00
0	0	1	Impossible	-	-
0	1	0	Impossible	-	-
0	1	1	Impossible	-	-
1	0	0	Zerotree Root	ZTR	01
1	0	1	Valued Zerotree Root	VZT	10
1	1	0	Value	VAL	11
1	1	1	Value	VAL	11

The following is a definition for the generation of the symbols

A ***DNT*** symbol is used to indicate a Do Not Transmit state within a pixel and, therefore, prevent any symbol transmission from this pixel to the receiver. This extra symbol is not considered a waste of bit-allocations as three symbols require two bits in any case. The ***DNT*** symbol is generated in a pixel when all three, the parent, the children and the pixel itself are insignificant. In cases where the pixel or its descendants are significant the parent (*Pin*) signal can never be logic 0. Therefore these conditions are considered impossible. In this case the *Pout* signal is held low and a coefficient is not expected from this pixel.

A ***ZTR*** symbol is generated by a pixel which is not significant, has no significant descendants but contains a significant parent (or sibling signalling via the parent). In this case the *Pout* signal is held low and a coefficient is not expected of this pixel.

A ***VZT*** symbol is generated by a pixel which is significant, yet contains no significant descendants. Pixels generating these symbols are expected to also transmit coefficients. In this case the *Pout* signal is held high during encode.

A ***VAL*** symbol is generated by a pixel that contains one or more significant descendants. Pixels generating these symbols are expected to provide a coefficient even if the coefficient is zero and is a dictated by the ZTE algorithm.

The two symbols ***VAL*** & ***VZT*** represent the key difference between the ZTE symbols and EZW symbols (***POS***, ***NEG*** & ***IZO***).

Given these conditions a possible architecture for the generation of these symbol bits (*ZS1* and *ZS2*) is presented in Figure 5-21.

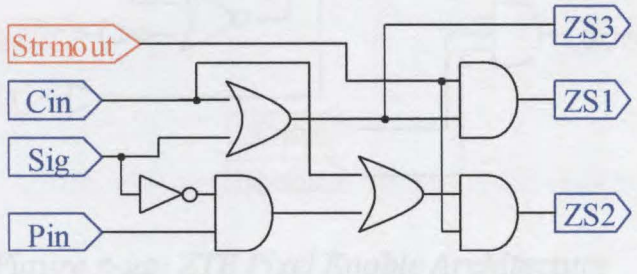


Figure 5-21: ZTE Classification Architecture

The *Strmout* signal is employed to clear the outputs during the decoding phase (*Strmout* = 0). The *ZS3* signal is used for pixel disabling and is placed here to reduce required components.

5.4.4. Pixel Enable Architecture

In addition to the scale control signals *VEnable* and *HEnable* the ZTE architecture also requires a means of disabling a pixel when it contains no data to transmit (i.e. assign *GEnable* to zero). The proposed pixel disable mechanism employed by ZTE architecture resembles that of the EZW because during the symbol encode and decode cycles the *Pin* signal controls the deactivation (*Pin* = 0) of pixels otherwise only controlled by the scale control architecture. This is because if a parent is insignificant then there is no reason for a child to transmit a symbol. When encoding and decoding the coefficient itself, the pixel disable, due to the zerotree component, is governed by the existence of a ***VAL*** or ***VZT*** symbol. These in turn, during the encode cycle, are generated via the logical OR operation performed on signals *Cin* and *Sig*. Since this function is performed previously in the symbol generation logic, the signal *ZS3* is employed to deliver the result here. Given this specification, the *GEnable* signal is derived by the architecture presented in Figure 5-22.

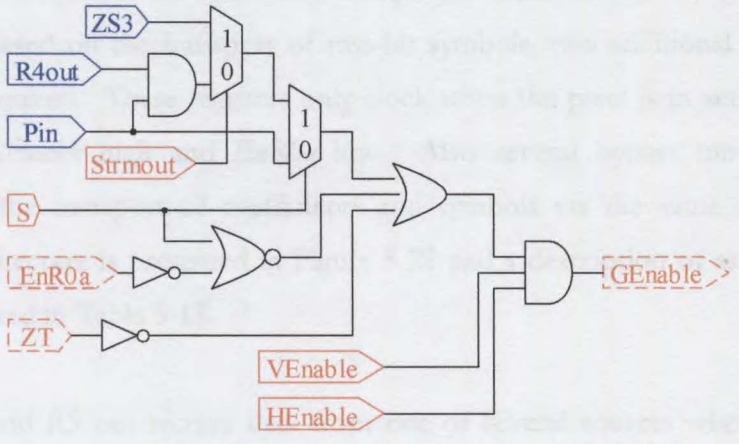


Figure 5-22: ZTE Pixel Enable Architecture

In zerotree mode (*ZT* = 1) the pixel is enabled if both *VEnable* and *HEnable* are high and one of the following conditions are true.

1. The *Pin* signal is high in both symbol encode and decode mode. This implies that the parent is significant or has significant descendants. Hence the pixel should be prepared to transmit or receive symbols.

2. The *ZS3* signal is high when encoding coefficients. This implies that the pixel is a Value or a Valued ZeroTree root, therefore, requiring coefficient transmission.

3. Both the signal *Pin* and the contents of register R4 are high, when performing coefficient decode. This implies a pixel's parent is significant and the recently received symbol is a Value or Valued ZeroTree root. Therefore, this pixel is enabled to receive a coefficient.

4. Finally, when *EnR0a* is high during symbol encode mode ($S = 0$). This allows cycling of R0 in zerotree mode before any significance statuses have been determined. During refinement passes ($S = 1$), however, this is disabled so that only significant pixels are held enabled.

5.4.5. Symbol Latching & Bypass Architecture

The pixel latching and bypass architecture provides for the capture and transport of symbols and coefficients to and from the pixel. Since the ZTE, like the EZW algorithm, is based on the transport of two-bit symbols, two additional registers (R4 and R5) are required. These registers only clock when the pixel is in zerotree symbol mode, with *GEnable* high and *EnR0a* low. Also several bypass mechanisms are proposed for the transport of coefficients and symbols via the same output. The proposed architecture is presented in Figure 5-23 and a description of associated I/O ports is presented in Table 5-13.

Registers R4 and R5 can receive data from one of several sources when in zerotree mode. These are listed as follows.

1. **Signals *ZS1* & *ZS2*** – These signals originate from the symbol generation circuitry providing a symbol in encode mode (*Strmout* = 1), and a zero value in decode mode

(*Strmout* = 0). The signals are captured when *LR1245* is set to logic 0 and the circuit is clocked. This function is used for encoding.

2. **The External Data Load Signal EXT** – When *LR1245* is set to logic 1 data originating at the EXT signal is loaded into register R4 and data at register R4 is load into register R5. The circuit acts as a large grouped (2-bit groups) shift register when several pixels are connected together.

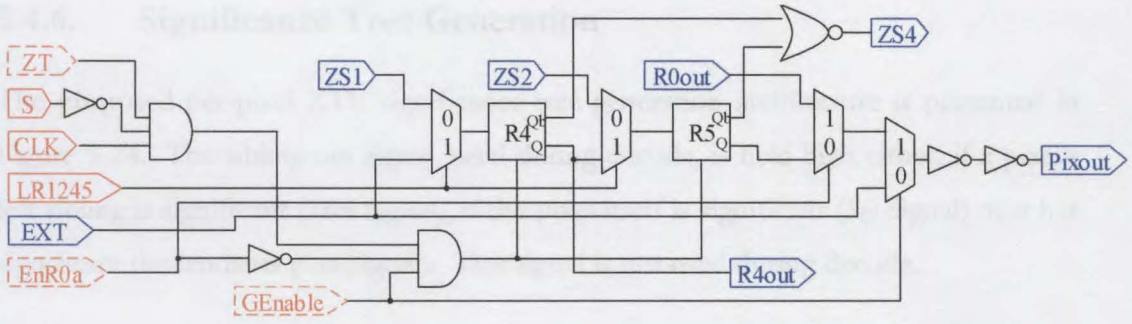


Figure 5-23: ZTE Latch & Bypass Architecture

Table 5-13: Latch & Bypass I/O Ports

Signal	Dir	Type	Orig/Dest	Description
CLK	In	Clock	External	Global Clock Signal
LR1245	In	Control	External	Select Load for R4 & R5
EnR0a	In	Control	Internal	Register R0 Clock Enable / ZT Bypass
GEnable	In	Control	Internal	Pixel-wise Enable
ZT	In	Control	Internal	Zerotree Mode Select
EXT	In	Data	External	Data From Surrounding Pixels
Pixout	Out	Data	External	Data To Surrounding Pixels
R0out	In	Data	Internal	Data From LSB of Register R0
R4out	Out	Data	Internal	Data For Pixel Enable in Decode
ZS1	In	Data	Internal	Symbol bit 1
ZS2	In	Data	Internal	Symbol bit 2
ZS4	Out	Data/Cont	Internal	Parent Significance During Decode

The architecture also consists of the following bypass modes.

1. **Pixel Bypass Mode** – If the *GEnable* signal is set low, then all data originating at *EXT* will be routed to *Pixout*. In this mode the pixel acts as a transparent pixel to its immediate neighbours.

2. **Symbol Bypass Mode** – When *EnR0a* is set high the pixel enters this mode. In this mode the data from register R0 is routed to *Pixout*. This mode is used to transmit data from register R0.

Finally the architecture also provides a signal, *ZS4*, which is used in the decode phase. This signal is held high when registers R4 and R5 contain a ***VAL*** symbol, indicating that its immediate children should expect symbols.

5.4.6. Significance Tree Generation

The proposed per-pixel ZTE significance tree generation architecture is presented in Figure 5-24. The sibling out signal, used during encode, is held high either, if a pixel's left sibling is significant (*Sbin* Signal), if the pixel itself is significant (*Sig* Signal) or it has significant descendants (*Cin* Signal). This signal is not used during decode.

During encode (*Strmout* = 1) the *Pout* signal is held high if a pixel has significant descendants, as this implies the generation of a ***VAL*** symbol. During decode (*Strmout* = 0) the *Pout* signal is held high if the contents of register R4 and R5 indicate a ***VAL*** (11b) symbol. This signal is provided by the *ZS4* signal. The *Pout* signal allows the descendants of a particular pixel to expect the arrival or transmission of a symbol and hence self-activate.

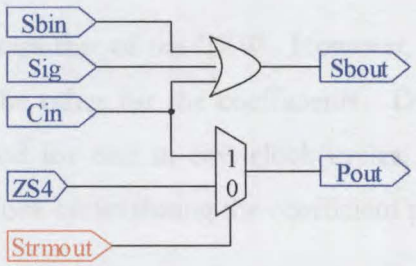


Figure 5-24: ZTE Significance Tree Generation

5.4.7. Data Extraction/Load Architecture

The data extraction and load architecture proposed for the ZTE is identical to that proposed for the EZW in Section 5.3.6. The only difference lies in the control

sequence. Instead of iteratively alternating between the symbol and refinement passes, the control structure here performs a single symbol and a single coefficient pass.

Initially, the scale control lines are set to enable only the lowest subband. Then in an encode cycle, when capturing the symbols into R4 and R5, the *FZT* signal is activated for one clock cycle also, so as to load *GEnable* into register R6. Then the signal *LR1245* is activated to link all the R6 registers into one array sized column shift register. Similarly the *RC* and *LR* signals are set to logic 1 and 0 respectively to connect the main pixel interconnects in a vertical shift direction from top to bottom. Then, for every symbol shifted (two clock cycles) the *FZT* signal is activated for one clock cycle to shift the counter bit. The arrival of a zero at the bottom pixel's *Below* output indicates that the column contains no more symbols to send. When all columns produce a zero the next subband can be activated and the symbol shift repeated for it. When all symbols have been extracted the *S* signal is activated for coefficient extraction mode. Again, the lowest subband of pixels is first activated. Then for one clock cycle the signals *LR1245* and *FZT* are set to logic 0 and 1 respectively to load *GEnable* into R6. The symbols do not reload as *S* is held high. The signals *LR1245* and *FZT* are then set to logic 1 and logic 0 respectively, also *EnR0* (not EnR0a, which is a derived signal) is set to logic 1. For each eight shifts (8 clock cycles) of the register R0, the *FZT* signal is held high for one, so as to shift the counter bit. As with the symbols, coefficient shifts stop when the bottom pixel produces a zero on its *Below* signal.

The decode mode also follows that of the EZW. However, only two passes are made, one for the symbols and the other for the coefficients. During the symbol pass the *FZT* signal is only activated for one in two clock cycles, while the same signals is activated for one in eight clock cycles during the coefficient pass.

5.4.8. Array Edge Buffer Design

Since the data extraction and load architecture contains virtually no difference between the ZTE and EZW algorithms, the edge buffers designs also follow suit. There are only two differences between the buffer designs.

1. **Register BR4 is extended to 8-bits** – Since the ZTE architecture can produce an 8-bit coefficient value instead of a single refinement bit as generated by the EZW architecture, register BR4 is extended to handle 8-bits instead. The register BR4 then becomes an 8-bit shift register.

2. **The control structure is modified to handle 8-bit cycles** – Since the pixel array control structure is modified to handle 8-bit coefficients the buffer control structures have to also be modified to perform the same. In addition the control structure has to be modified to handle only two passes instead of several alternate passes in the EZW architecture.

These changes only present a simple set of modifications and as such do not require explicitly defined architecture.

5.4.9. The Encode Control Sequence

The full ZTE encode cycle for either an array or single pixel structure is presented in Table 5-14. The variable X is used to represent the number of quantisation cycles. It is dependant on the user selectable subband quantisation mechanism and the chosen quantisation coefficients, such as that described in Sections 2.6.2, 3.3.1 and 4.6.5.

5.4.10. The Decode Control Sequence

The full ZTE decode cycle for either an array or single pixel structure is presented in Table 5-15. The variable Y is used to represent the number of de-quantisation cycles. It is dependant on the user selectable subband quantisation mechanism and the chosen quantisation coefficients, such as that described in Sections 2.6.2, 3.3.1 and 4.6.5.

Table 5-14: ZTE Encode Cycle

Phase	Description	M1	M2	R0Ct1	R0Ct2	LR	RC	S	EnR0	Strmout	LR1245	Rev	Fzt	ClkMSB			ClockCycles
1	**Select a Subband For Quantisation**																
2	Quantise Subband	1	0	0	0	0	0	0	1	0	0	0	0	0			X
3	**Repeat Phases 1 & 2 Until all Relevant Subbands Have Been Quantised**																
4	**Enable Entire Array**																
5	Significance Detect	1	1	0	0	0	1	0	1	1	0	0	0	0			8
6	**Enable Lowest Frequency Subband**																
7	Latch Symbols & R6	1	1	0	0	0	1	0	0	1	0	0	1	0			1
8	Extract R5	1	1	0	0	0	1	0	0	1	1	0	0	0			1
9	Extract R5 & R6	1	1	0	0	0	1	0	0	1	1	0	1	0			1
10	**Repeat Phases 8 & 9 Until Bottom Pixel displays 0 in Below**																
11	**Repeat Phases 7 - 10 For all Subbands**																
12	**Enable Lowest Frequency Subband**																
13	Latch R6 for Coeff	1	1	0	0	0	1	1	0	1	0	0	1	0			1
14	Ex. R0 7 LSBs	1	1	0	0	0	1	1	1	1	1	0	0	0			7
15	Ex. R0 MSB & R6	1	1	0	0	0	1	1	1	1	1	0	1	0			1
16	**Repeat Phases 14 & 15 Until Bottom Pixel displays 0 in Below**																
17	**Repeat Phases 13 - 16 For all Subbands**																

Table 5-15: ZTE Decode Cycle

Phase	Description	M1	M2	R0Ct1	R0Ct2	LR	RC	S	EnR0	Strmout	LR1245	Rev	Fzt	ClkMSB			ClockCycles
1	**Enable Lowest Frequency Subband**																
2	Latch R6	1	1	0	0	0	1	0	0	0	0	0	1	0			1
3	Load R5	1	1	0	0	0	1	0	0	0	1	0	0	0			1
4	Load R5 & R6	1	1	0	0	0	1	0	0	0	1	0	1	0			1
5	**Repeat Phases 3 & 4 Until Top Pixel Displays 0 in Above**																
6	**Repeat Phases 2 - 5 For all Subbands**																
7	**Enable Lowest Frequency Subband**																
8	Latch R6 for Coeff	1	1	0	0	0	1	1	0	1	0	0	1	0			1
9	Ld. R0 7 LSBs	1	1	0	0	0	1	1	1	0	1	0	0	0			7
10	Ld. R0 MSB & R6	1	1	0	0	0	1	1	1	0	1	0	1	0			1
11	**Repeat Phases 9 & 10 Until Top Pixel Displays 0 in Above**																
12	**Repeat Phases 8 - 11 For all Subbands**																
13	**Select a Subband For De-Quantisation**																
14	De-Quantise Subband	1	0	0	0	0	0	0	1	0	0	0	0	1			Y
15	**Repeat Phases 13 & 14 Until all Relevant Subbands are De-Quantised**																

5.5. Codec Comparison

To aid in the decision to select a single codec for hardware prototyping, both the EZW and ZTE codecs are compared in this section.

5.5.1. Video Compression Performance

To provide a fair comparison, both zerotree codecs were tested using the same Motion Compensation, Triangular Wavelet Transform and Arithmetic Coder components (Chapter 4). Two sequences, the proprietary Jenny sequence and the standard Salesman sequence, were compared at two different bit rates, 64 Kbps (ISDN) and 250 Kbps (3G). More information on these two video sequences can be found in Section 2.2.4.

At 64 Kbps, both video sequences were sub-sampled from 25 fps to 5 fps to maintain a low bit-rate. Figure 5-25 illustrates the compression performance of the two algorithms on the Jenny sequence at 64 Kbps and at 5 fps. Figure 5-26 illustrates the same comparison using the salesman sequence. The Jenny sequence seems to show near identical performance between the two algorithms. However, a slight decrease in the uniformity of the PSNR in the EZW is noted. The ZTE preforms slightly better when compressing the salesman sequence, and also indicating a uniform curve.

At 250Kbps (~3G), both video sequences were sampled at 25 fps. Figure 5-27 illustrates the performance for the Jenny Sequence, while Figure 5-28 the Salesman sequence. The Jenny sequence seems to show somewhat similar performance, between the two codecs, while the Salesman sequence favours the EZW.

This suggests that at high bit-rates, for the sequence without a moving camera (Salesman), the EZW performs better, yet at low bit-rates the ZTE outperforms the EZW. For a moving camera sequence (Jenny) the performance difference between the two are marginal at both bit-rates. However, in all cases the ZTE seems to deliver the most uniform performance characteristic.

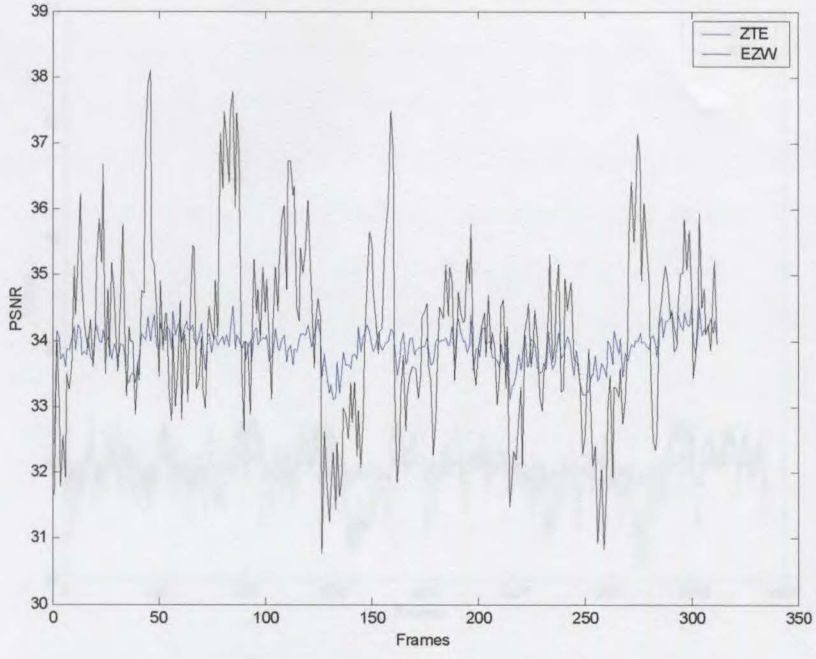


Figure 5-25: Jenny Sequence at 64Kbps (5fps)

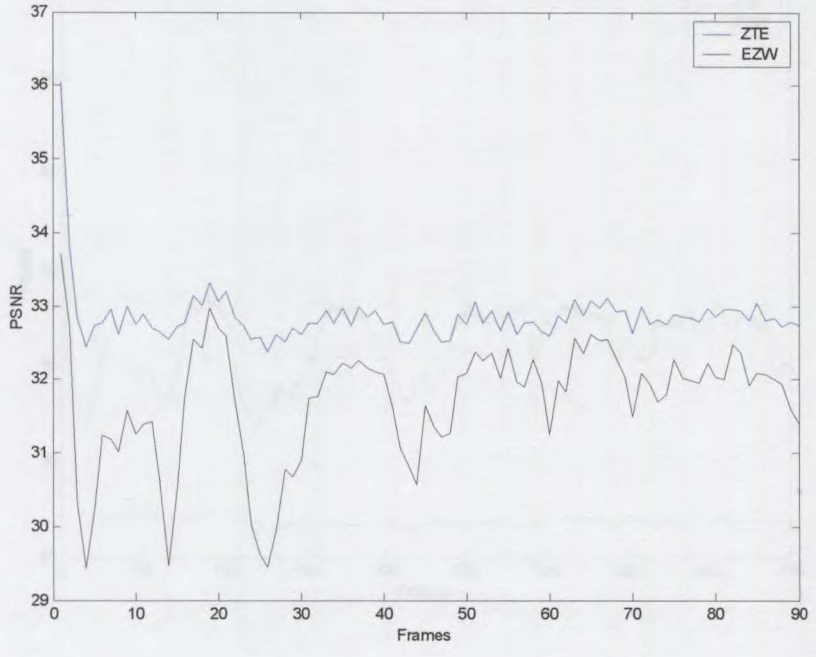


Figure 5-26: Salesman Sequence at 64 Kbps (5fps)

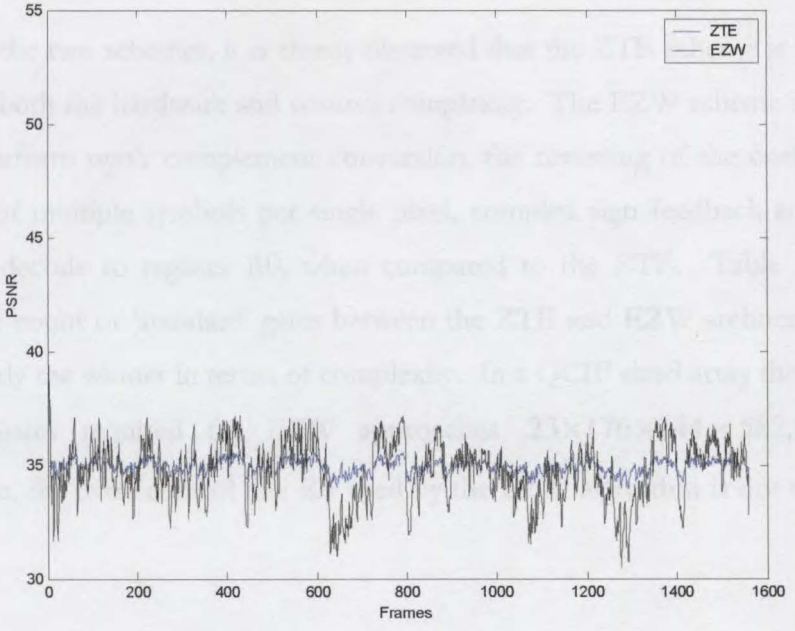


Figure 5-27: Jenny Sequence at 250 Kbps (25fps)

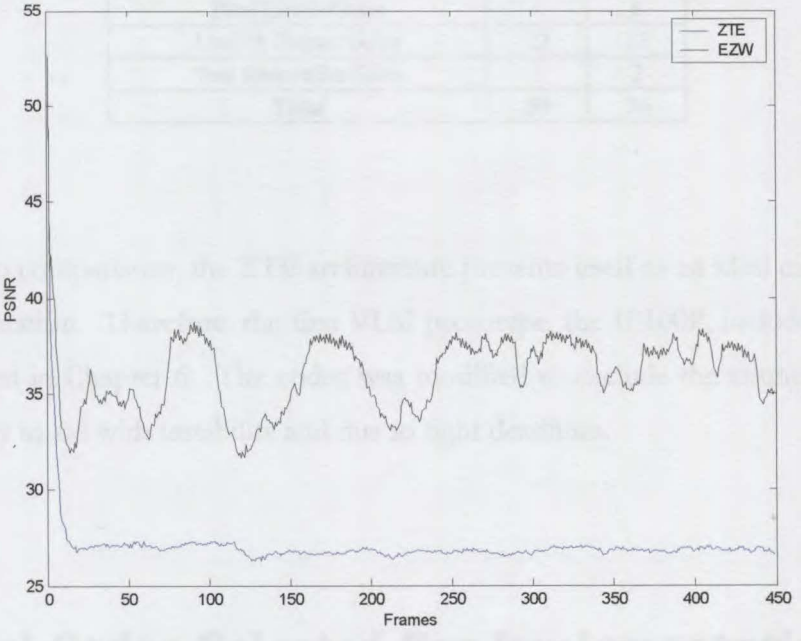


Figure 5-28: Salesman Sequence at 250 Kbps (25fps)

5.5.2. Hardware and Control Complexity

Comparing the two schemes, it is clearly observed that the ZTE scheme is far simpler, in terms of both the hardware and control complexity. The EZW scheme requires the ability to perform two's complement conversion, the reversing of the coefficient, the generation of multiple symbols per single pixel, complex sign feedback and complex refinement decode to register R0, when compared to the ZTE. Table 5-16 lists a comparative count of 'standard' gates between the ZTE and EZW architectures. The ZTE is clearly the winner in terms of complexity. In a QCIF sized array the number of additional gates required for EZW approaches $23 \times 176 \times 144 = 582{,}912$ gates. Furthermore, the extra control line *Rev* used by the EZW algorithm is not required for the ZTE.

Table 5-16: EZW vs. ZTE Complexity Comparison

Component	EZW	ZTE
Register R0 Multiplexes	12	5
Significance ID Gates	12	2
Self-Classification Gates	6	6
Pixel Enable Gates	4	8
Load & Extract Gates	22	13
Tree Generation Gates	3	2
Total	**59**	**36**

Given these two comparisons, the ZTE architecture presents itself as an ideal candidate for VLSI implementation. Therefore, the first VLSI prototype, the IP100P, includes the ZTE codec as detailed in Chapter 6. The codec was modified to exclude the extraction / load counter circuitry to aid with testability and due to tight deadlines.

5.6. Final Codec Selected For Implementation

The IP100P is the first prototype designed to test the functionality and power characteristics of such a low clock speed massively parallel compression codec. The design is a 32 x 32 array, which includes motion differencing, the wavelet transform, quantisation and ZTE coding. Since the author had not redesigned the wavelet transform circuitry at

the time of chip design, the author was forced to alter the ZTE design to allow it to integrate into the existing control structure present at the time. Although, the control signals were modified to better suit both components the functionality of the circuit still remains the same. Furthermore, since complementary CMOS VLSI circuitry optimisations favour the use of NAND and NOR gates over the traditional AND and OR gates, the ZTE architecture was further modified to include this optimisation. The symbol bits were also inverted to aid in this optimisation. The final ZTE design is presented in Figure 5-29 and the final wavelet transform / motion compensation architecture is presented in Figure 5-30. Some control signals such as *ZT*, *WT* and *MC* were modified to directly drive the individual components via the signals *SymIO* and *En_R1R2*. These provide equivalent functionality to *M1* and *M2* in the proposed design.

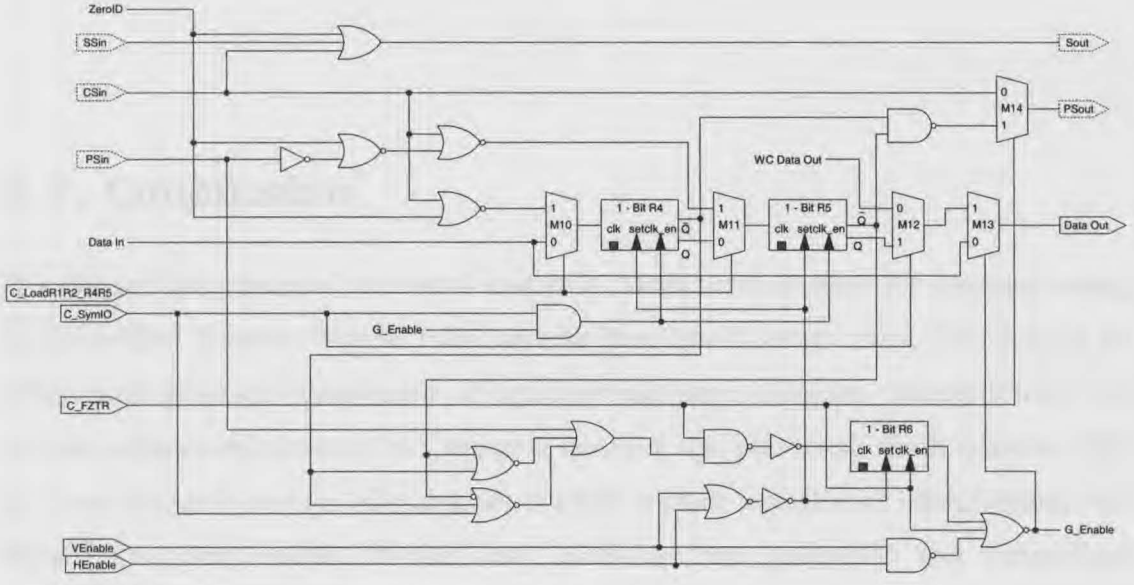


Figure 5-29; Final ZTE VLSI Schematic

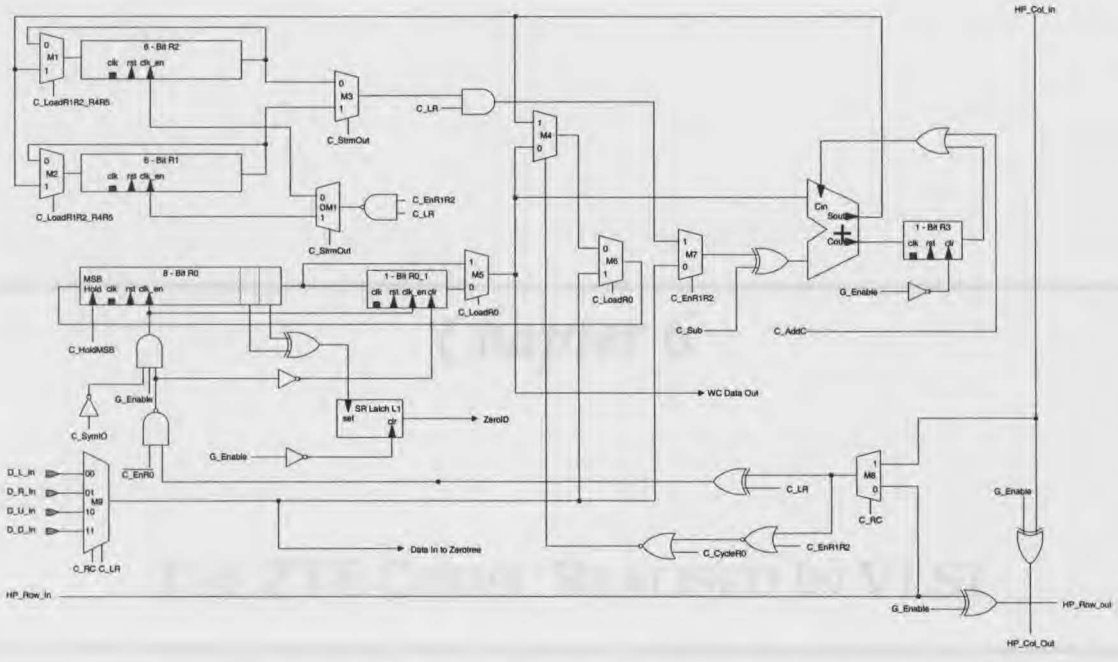


Figure 5-30: Final VLSI WT and MC Schematics

5.7. Conclusion

This chapter has presented two novel massively parallel architectures for zerotree coding; the Embedded Zerotree Wavelet coder and the ZeroTree Entropy coder. The designs are based on per-pixel self-classification of significant wavelet coefficients. Integration into the wavelet architecture, presented in Chapter 4, has been also presented. Each zerotree coder has been designed as a set of components which include significance identification, self-classification, pixel enable, Extract/Load, pixel-wise tree generation and extract/load counting architectures. A description of the functional requirements for the array-edge buffers has also been provided. Finally a decision for the use of the ZTE architecture in the design of the IP100P prototype has been established, due to limited space requirements, tight timelines and less complexity. As a result, the modified ZTE architecture used in the IP100P prototype and the Wavelet Transform / Motion Compensation architecture have also been shown.

Chapter 6

THE ZTE CODEC REALISED IN VLSI

"Neo, no one has ever done anything like this." - ***Trinity***

"That's why it's going to work." - ***Neo***

Matrix: The Movie

6.1. Introduction

In this chapter the feasibility of VLSI hardware implementation of a novel massively parallel Zerotree Entropy Coder, as introduced in the previous chapter, is explored. The work conducted herein contributes to the VLSI hardware realisation of the project code named IP100. The IP100 project, which originated as an idea in 1995, is currently composed of three fundamental streams of research, segmented by function. They are...

1. **The IP100C** – The capture component researched and developed at Edith Cowan University and University of Ulm.
2. **The IP100D** – The display component researched and developed at the University of Cambridge.
3. **The IP100P** – The processing component researched and developed at Edith Cowan University and University of Las Palmas de Gran Canaria.

The author was primarily responsible for the development and implementation of the ZTR components belonging to the IP100P prototype. As such this chapter highlights the key

ZTE components and the other related support components developed for the IP100P by the author.

6.2. IP100P Prototype Configuration

The IP100P's primary function is to demonstrate the design and implementation feasibility of a novel pixel based video compression approach in current VLSI technology. As such, and to aid with the testing the chip design is limited to the implementation of the array only. Such components as array edge buffers; arithmetic coders, stream coders, control logic etc. are omitted for this design. Figure 6-1 illustrates the I/O and control pin allocation strategy for the realisation of a 32 x 32 IP prototype array. 32 switchable Input / Output columns allow for array load and unload while 18 control lines provide for a versatile external control mechanism.

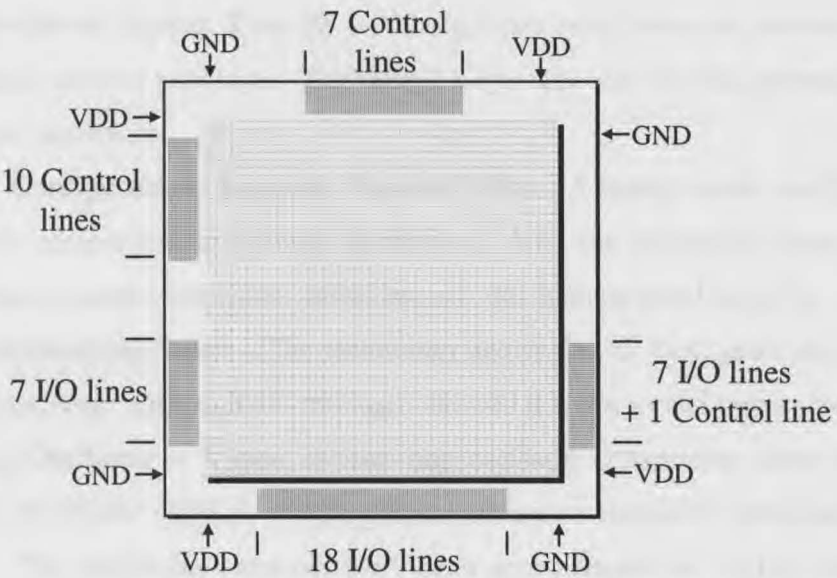


Figure 6-1: IP100P Prototype I/O Configuration

This I/O pattern was chosen to suit the per-side pin-count of a JLCC 84 pin chip carrier.

6.3. The Technology

The technology employed for the realisation of the IP100P and its sister prototype, the IP100C, present the following features.

1. **Foundry: UMC (United Microelectronics Corporation)** – Chosen because of ease of access via Europractice and because of financial and design support merits.
2. **Minimum feature size: 0.25 micron** – At this minimum size an array approximating to 16 x 16 mm^2 easily supports a QCIF image with adequate yield. The minimum transistor size W = 0.3μm, L = 0.24μm.
3. **Power Supply Configuration: 2.5V & 3.3V** – The pads require a 3.3V (VCC) I/O signal, however the internal circuits operate on 2.5V (VDD). A binary *'off'* is represented by a 0V signal, while a binary *'on'* is represented by a 2.5V signal.
4. **Well Configuration: Twin Well** – This process employs a Twin-Well process with the substrate acting as the well for the NMOS transistors and a P+ implant applied inside an N-Well for the PMOS transistors.
5. **Metal Layers: Five (5)** - This caters for the use of four metal layers for routing and interconnects and one layer (5) for the polished mirror driver. Metal 5 was not used as a functional layer for the IP100P design.
6. **Poly-silicon Layers: Two (2)** – Although two poly layers are present the design utilised only one poly layer. The second is typically used for the generation of poly-silicon capacitors.
7. **VIA Configuration: Supports Stacked VIAs** – A highly useful configuration for circuit compaction and power distribution. Vias can be stacked from the lowest diffusion contact through all metal layers to the highest metal layer (5)
8. **45^0 Manhattan Rules** – The technology allows for 45^0 bent gates and routing for compactness. Although 90^0 routing is allowed it is not a valid option for the gates.
9. **Ring Oscillator** – A nine inverter ring oscillator constructed from the supplied level 49 BSIM3 NMOS, PMOS HSpice transistor models is presented in Figure 6-2. This oscillating frequency for 9 gates approximates to 1 GHz, which implies that the maximum operating frequency of a single minimum size gate can be calculated from $1 \times 9 / 1n\sec$ = 9 GHz. This is also confirmed by evaluating ring oscillators with 3, 5, and 7 inverters. Therefore the propagation delay for each cascading minimum size gate is 1/9 GHz = 0.111 ns. As a result if 60,000 gates are connected in series then at 100KHz the propagation delayed applied by these gates will result in significant clock skew. The average power consumption for these 9 gates operating at 9 GHz is approximately 500μW

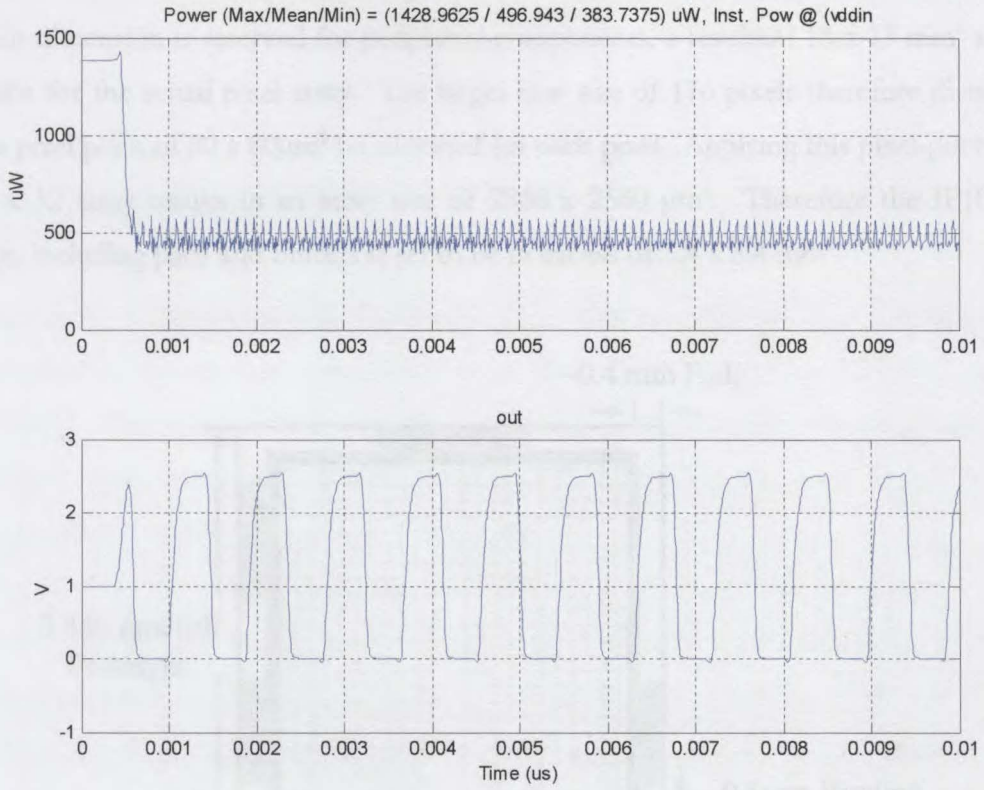


Figure 6-2: Ring Oscillation for UMC 0.25µm Process

The full custom layout work and simulation of the IP100P prototype in UMC 0.25 micron technology was completed in approximately four months, with the use of tools including Cadence (Virtuoso, Dracula, Analog Artist), HSpice, Electric, Synopsis and VHDLSimili.

6.4. IP100P Floor Plan

A 32 x 32 array comprised of 1024 individual pixel processors each with identical content, which allowed for a hierarchical instance based layouts to be employed for the IP100P realisation. This section will cover some of the top-level layout considerations associated with the layout of the IP100P.

6.4.1. IP100P Size & Layout Considerations

The size of the full chip is easily derived from the size of a QCIF (176 x 144 pixel) image, as this is the targeted size for the final product. If a size restriction of 16 x 16

mm^2 is to be maintained due to cost, yield and technology factors, and a further 3 mm in each dimension is reserved for peripheral components, a resultant 13 x 13 mm^2 area remains for the actual pixel array. The larger row size of 176 pixels therefore dictates that a pixel pitch of 80 x 80μm^2 be allocated for each pixel. Applying this pixel-pitch to a 32 x 32 array results in an array size of 2560 x 2560 μm^2. Therefore the IP100P design, including pads and buffers is set to be in excess of 3.4 x 3.4 mm^2.

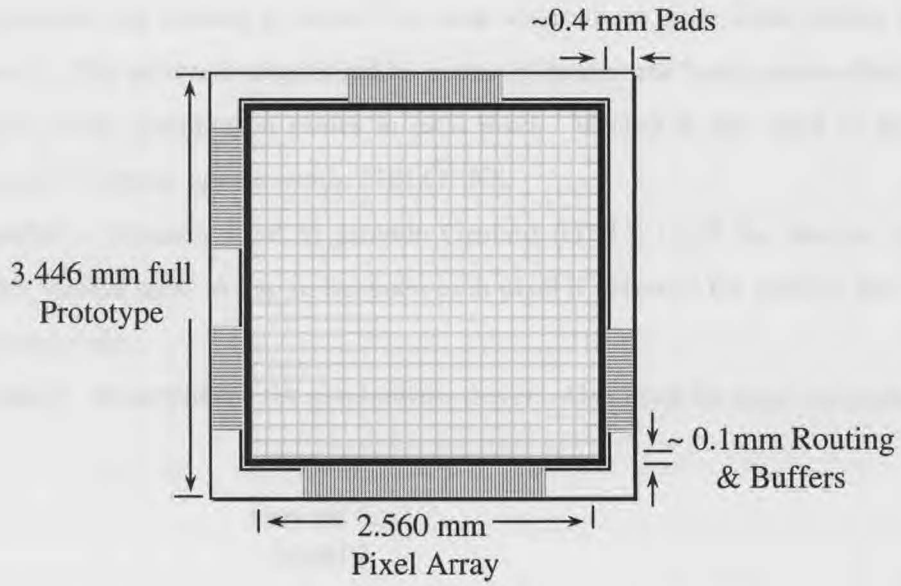


Figure 6-3: IP100P Proposed Dimensions

6.4.2. Power Distribution & Metal Allocation

The power distribution strategy used in a design is generally related to the number of metal layers available. A five metal process was selected as it offered a good compromise between the required reflectivity of the top metal layer and the required resolution for use as the LCD SLM. In addition, when selecting the number of metal layers, the photodetector at the substrate level demands some special consideration. Since the photodetector area is to be left uncovered for capture of light, having many metal layers requires a highly refined cut creation process, increasing the chance of creating defective photodetectors. Furthermore, if the depth of the well in which the photodetector resides is not kept to a minimum the shadows cast by the top metal layers can interfere with the capture ability of the photodetector. Taking this into

account along with the fact that metal five is reserved for the SLM, the allocation of the available metallisation layers is presented below.

1. **Metal-1** – Used in the design of primitive components and some higher level routing when possible.
2. **Metal-2** – Used for interconnects between primitive components.
3. **Metal-3** – Primarily used to distribute VDD to all the devices. It is also used to supplement the routing in metal-2 to areas where there is no viable access through metal-2. The power is distributed by means of horizontal bands across the array to major power distribution nodes in each pixel. Metal-3 is also used to distribute some ZTE Status signals within NBLOCKS.
4. **Metal-4** – Primarily used to provide Ground (GND) to all the devices via pixel width vertical rails. A gap in between each pixel is reserved for control line routing in metal-4 also.
5. **Metal-5** – Reserved for the pixel mirror driver. Also used for logo and pads.

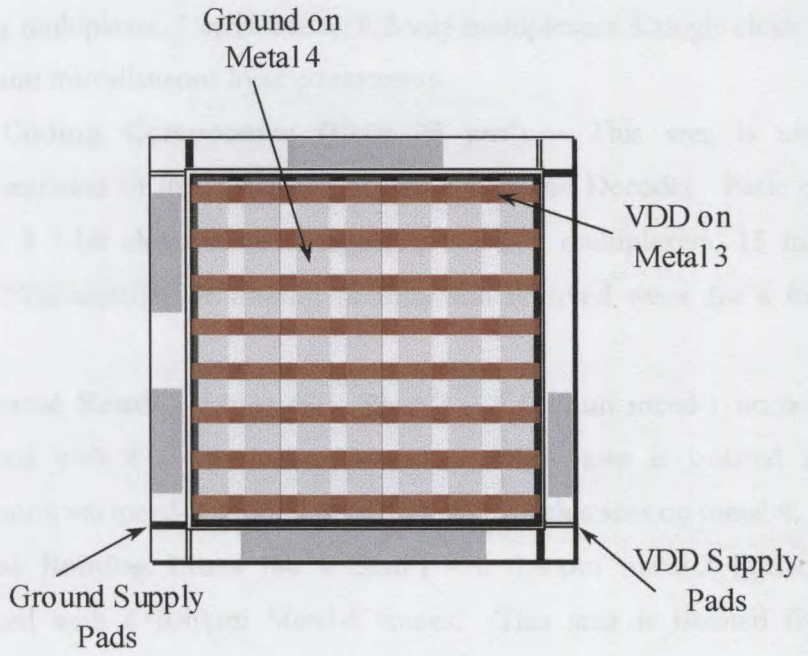


Figure 6-4: Power Distribution in IP100P Prototype

Figure 6-4 illustrates the power and ground distribution methodology for the supply of power throughout the array. Pads placed at the vertices of the array provide for the delivery of external power into the array. A second potential of 3.3V is required for

pad interfacing and is supplied with each block of control or I/O pads through the use of a VCC pad.

6.4.3. Pixel Floor-Plan

The area for each pixel sub-divides into 7 ({1} – {7}) distinct zones, as shown in Figure 6-5. These areas are reserved for the following functions.

1. **Photodiode (15 x 15 μm^2)** – This area is reserved for the implementation of a photodiode device in future versions.
2. **Analogue to Digital Converter (15 x 60 μm^2)** – This area is reserved for the implementation of an Analogue to Digital Converter to accompany the photodiode in future versions.
3. **Wavelet Coding & Motion Differencing (50 x 75 μm^2)** – This area is used for the realisation of the wavelet coding and motion compensation circuitry. Some basic components in this area include – 1 9-bit shift register, 2 6-bit shift registers, 1 4-way multiplexer, 1 serial adder, 9 2-way multiplexers, a single clock distribution driver and miscellaneous logic components.
4. **ZTE Coding Components (10 x 75 μm^2)** – This area is used for the implementation of the Zerotree Entropy Coder and Decoder. Basic components include 3 1-bit clearable shift registers, 5 2-way multiplexers, 13 miscellaneous gates, ZTE significance routing circuitry and reserved space for a future mirror driver.
5. **Horizontal Routing Lines (5 x 80μm^2)** – 6 0.40μm metal-1 horizontal routes interlaced with 6 0.40μm Metal-3 routes. This area is isolated from VDD distribution via metal-3, ground is distributed over this area on metal-4.
6. **Vertical Routing Lines (80 x 5μm^2)** - 6 0.40μm metal-2 horizontal routes interlaced with 6 0.40μm Metal-4 routes. This area is isolated from ground distribution via metal-4, VDD is distributed over this area via metal-3. Furthermore other metal-3 routes may be utilised where VDD distribution is not present (i.e. ZTE Status lines).
7. **Vertical Centre Routes (80 x 5μm^2)** – This zone is kept free of metal-2 when possible so as to allow signal distribution via the centre of the pixel.

With these floor plans the layout of the NBLOCK and hence the array can commence.

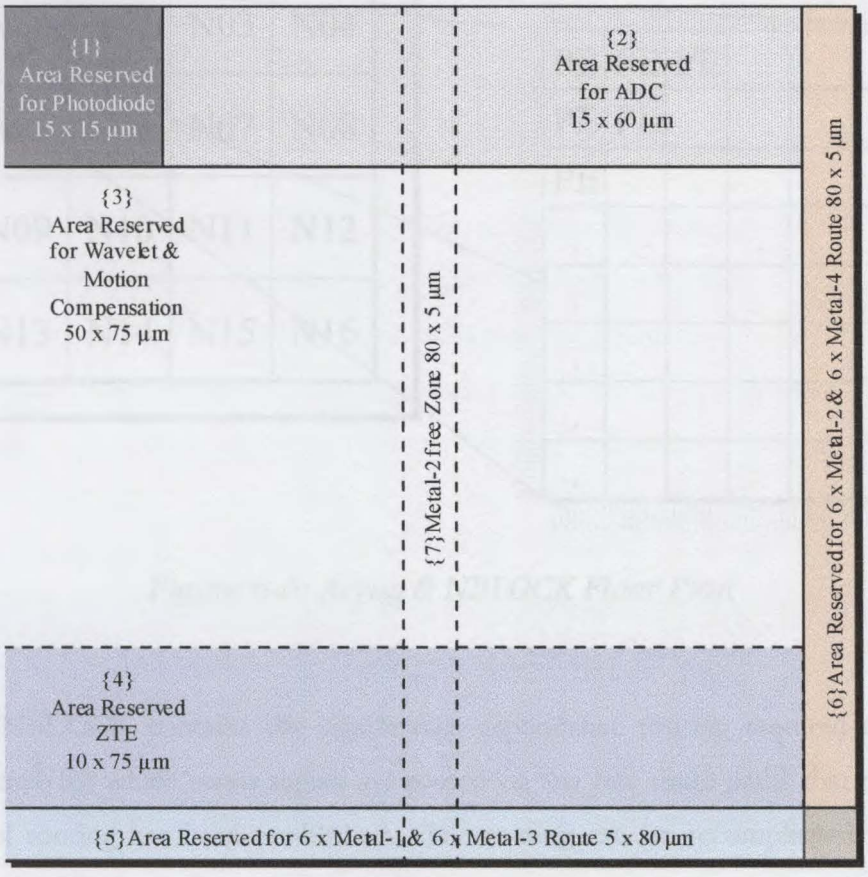


Figure 6-5: Pixel Floor-Plan

6.4.4. NBLOCK Arrangement

Nucleic Blocks (NBLOCKS) provide the smallest individual scalable component in the array. A useful result of the approach is the novel ability to scale the size of the array to suit any integer multiple of NBLOCKS. For a 3 scale DWT the size of the ZTE dependence tree results in an NBLOCK size of 8 x 8 pixels. Routing, especially with regards to the ZTE Status components, is unique within an 8 x 8 block of pixels. In the case of the IP100P prototype, a set of 4 x 4 NBLOCKS are used to construct an array of 32 x 32 pixels, as illustrated in Figure 6-6.

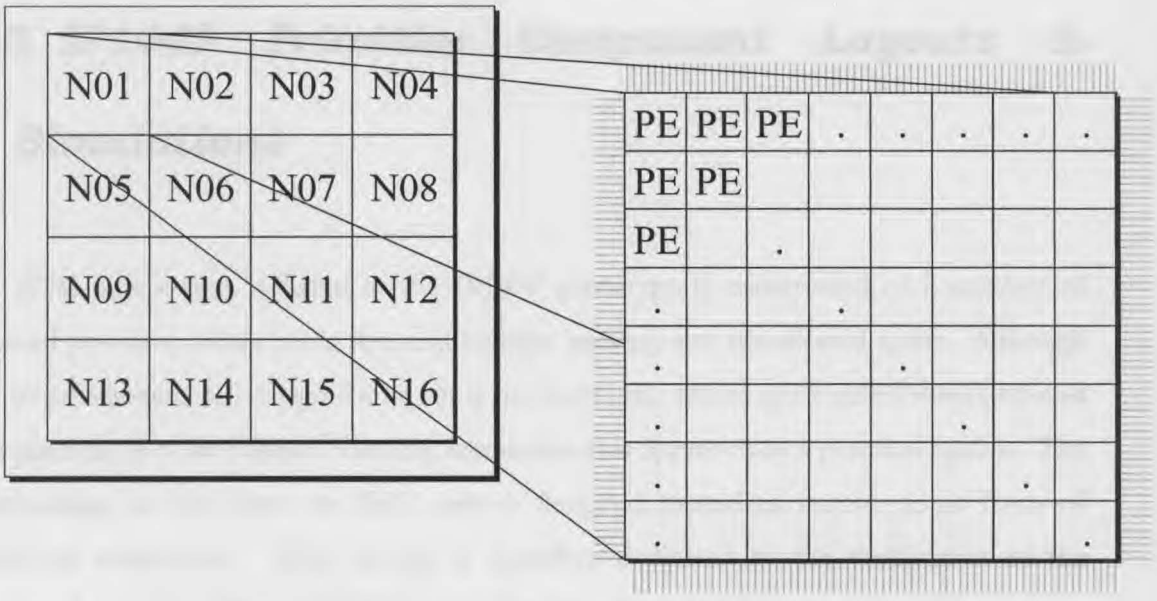


Figure 6-6: Array & NBLOCK Floor Plan

Each NBLOCK contains the significance dependence routing required for a single ZTE tree, for which status signals are routed via any free route paths that remain after control routing has been established. The routing can be accomplished using 2 - 5 vertical and horizontal routes per pixel.

The IP100P prototype design, after control routing is placed, provides for 5 free vertical and 3 free horizontal routes. These routes are utilised to perform the ZTE significance routing.

6.4.5. Control Signal Routing & Buffer Placement

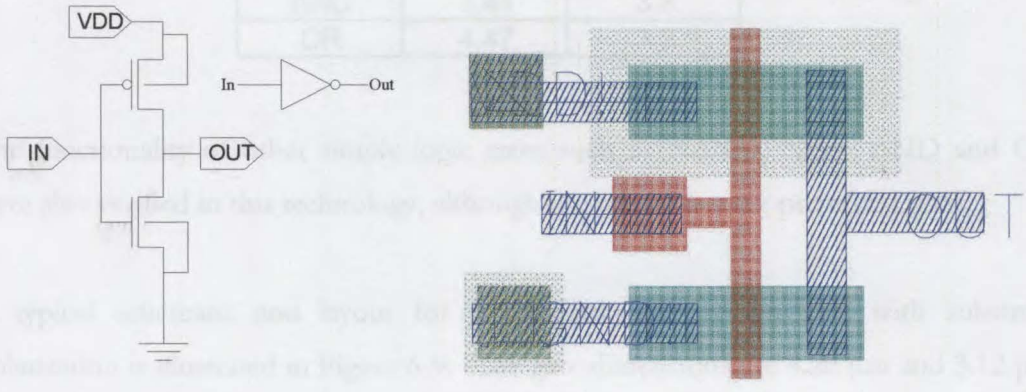
All control signals are routed via either a set of horizontal or vertical control lines. Generally all control lines propagate to all pixels in the array, with the exception of the scale control lines that only propagate to appropriate pixels belonging to a particular scale. The buffers that drive these global control lines are distributed at the borders of the IP100P layout. To accommodate for two different transistor drive loads, two inverter buffer stage designs are used, which are presented in the next sections. These buffers are attached to the appropriate horizontal or vertical control lines for signal delivery to the pixels.

6.5. IP100P Primitive Component Layouts & Simulations

The ZTE component included in the IP100P prototype is constructed of a number of essential primitive components that require tight packing into the allotted space. Although the use of standard cell design for layout is the norm, the severe space restrictions imposed by a pixel-pitch of 80 x 80μm^2 virtually illuminates this approach as a practical option. The disadvantage is that these the fully custom designed primitives require some form of functional verification. This section is therefore dedicated to the verification of the functional aspects of these primitive components. It should be noted that the primitive designs used herein is also employed in the wavelet component and as such were designed and verified as a group effort. The designs investigated here are adopted from [68].

6.5.1. Basic Primitives

A typical schematic representation for a CMOS inverter together with its equivalent minimum width layout in UMC 0.25 micron technology is presented in Figure 6-7. The operation of the inverter and the power consumed when driving a typical single minimum size gate is presented in Figure 6-8. The dimensions in terms of width and height for this inverter and other basic logic gates are presented in Table 6-1.



(a) (b)

Figure 6-7: Inverter Schematics & Layout

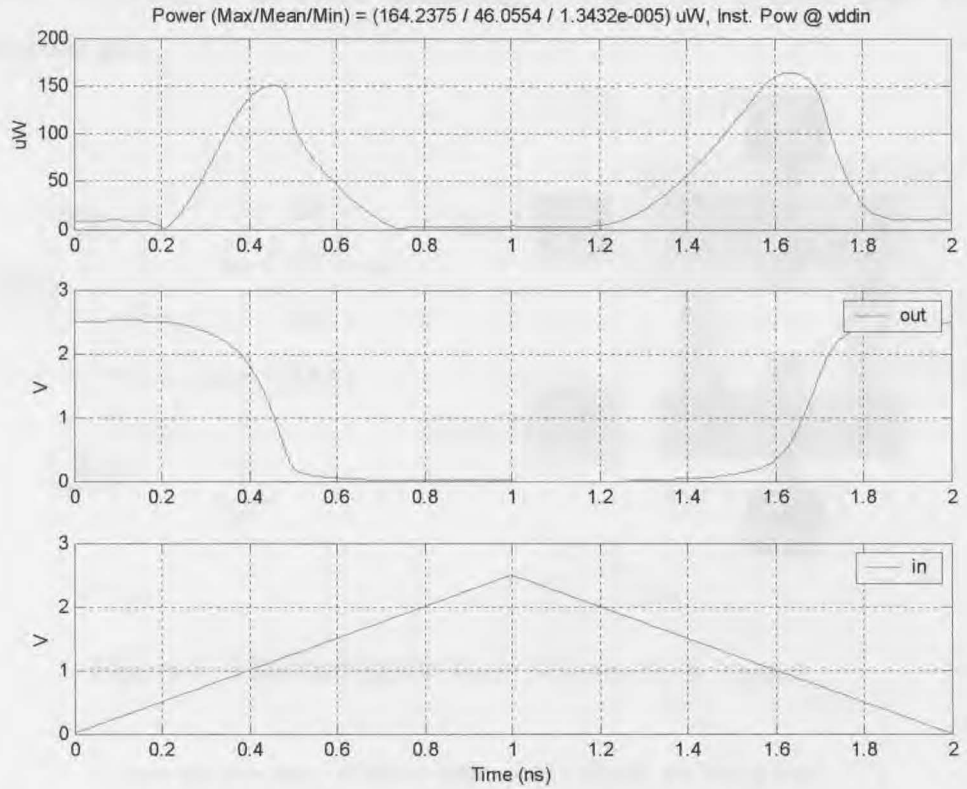


Figure 6-8: Inverter Operation

Table 6-1: Primitive Gate Dimensions

Gate	Width (µm)	Height (µm)
INV	3.12	3.3
NAND	4.92	3.3
NOR	3.88	3.3
AND	3.44	3.3
OR	4.47	3.3

The functionality of other simple logic gates such as NAND, NOR, AND and OR were also verified in this technology, although the results are not presented here.

A typical schematic and layout for a standard transmission gate with substrate polarisation is illustrated in Figure 6-9. The gate dimensions are 4.20 µm and 3.12 µm for the height and width respectively. The gate acts as a link when $\overline{Sel}$ is grounded and a VDD signal is applied to Sel. When the reverse is applied both MOSFETS fail to make the threshold voltage V_T, irrespective of the input or output levels. Figure 6-10

graphs the operation of this transmission gate. Before the *Sel* signal is set at 2.5ns, an unknown condition is observed indicating a typical disconnected state for a transmission gate.

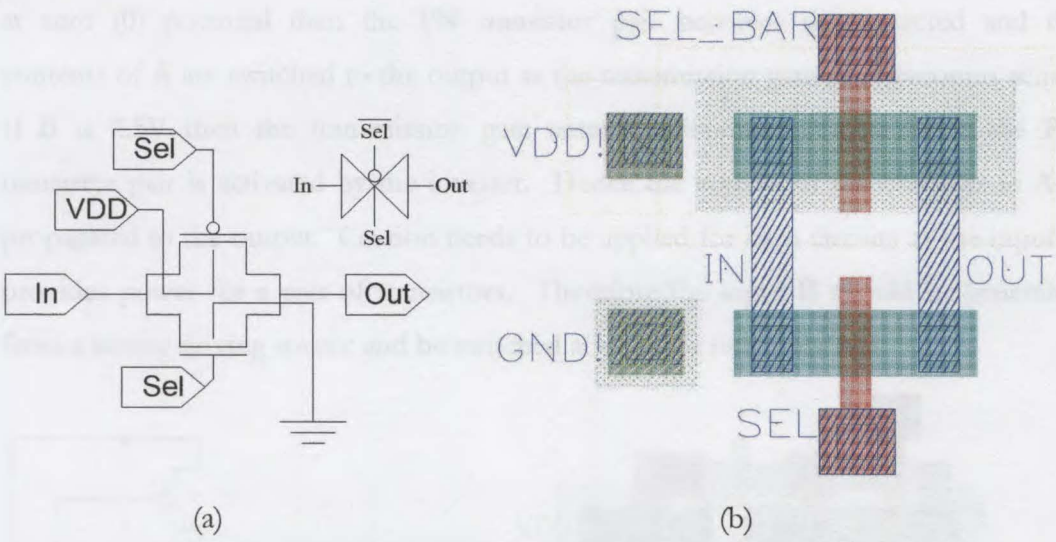


Figure 6-9 Transmission Gate Schematic & layout

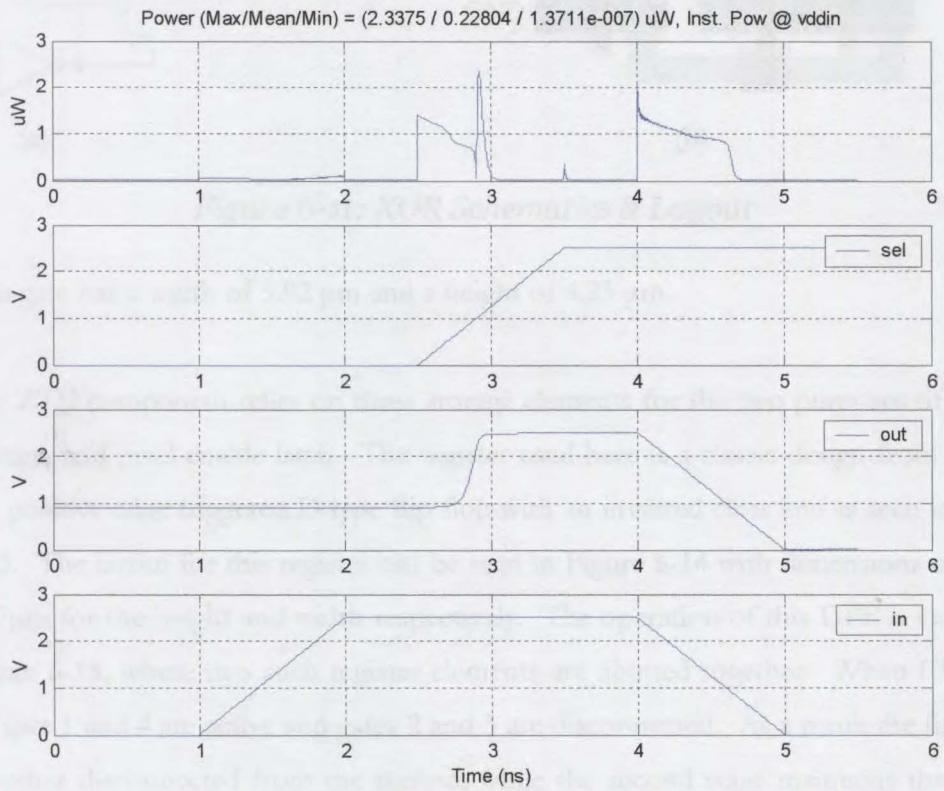


Figure 6-10: Transmission Gate Operation

The Exclusive OR gate, in the interest of saving space, has been implemented in a less stable manner as compared to typical logic design. It operates on the principle of a switchable inverter as is observed when studying the schematics in Figure 6-11. If **B** is at zero (0) potential then the PN transistor pair becomes disconnected and the contents of **A** are switched to the output as the transmission gate now becomes active. If **B** is 2.5V then the transmission gate enter a disconnected state while the PN transistor pair is activated by the inverter. Hence the inverse of the contents in **A** is propagated to the output. Caution needs to be applied for such circuits as the input **B** provides power for a pair of transistors. Therefore the input **B** should be generated from a strong driving source and be switched at a slower rate. [69] [70]

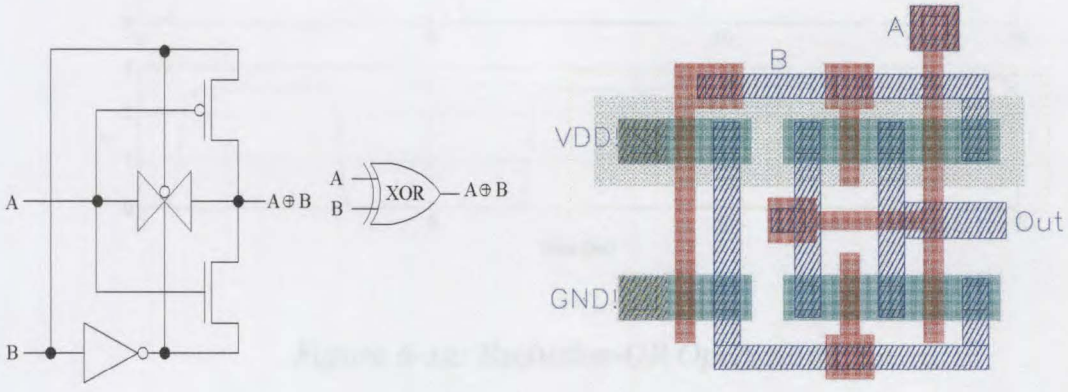


Figure 6-11: XOR Schematics & Layout

This gate has a width of 5.92 μm and a height of 4.23 μm.

The ZTE component relies on three storage elements for the two purposes of symbol storage, and pixel enable latch. The register used here is a classic design from [68]. It is a positive edge triggered D type flip flop with an inverted clear line as seen in Figure 6-13. The layout for this register can be seen in Figure 6-14 with dimensions of 8.80 x 9.07μm for the height and width respectively. The operation of this DFF is verified in Figure 6-15, where two such register elements are abutted together. When CLK is at 0V gate 1 and 4 are active and gates 2 and 3 are disconnected. As a result the first stage becomes disconnected from the second, while the second stage maintains the data at DOUT. When CLK is VDD (2.5V), gates 1 and 4 are disconnected the signal latched in the first stage is transferred to DOUT. The interesting event occurs when CLK rises from 0V to VDD.

Figure 6-12: Exclusive-OR Operation

Here the signal at DIN is latched and passed out to DOUT via a 4 gate delay. If two elements are cascaded then due to the delay of 4 gates the previous DOUT signal of register element 1 is latched to element 2 prior to this DOUT changing. When a downward transition of the CLK occurs, stage 2 of the element latches the signal at C and stage 1 is setup to DIN. This process results in the generation of a single clock cycle delay. The main advantage this register exhibits is the reduction of at least 10 transistors when compared to a standard DFF. Theis register only requires 20 transistors, which can result in a saving of up to 3.8 million transistors in a QCIF array. However care must be enforced to provide a "clean" clock signal as this register implementation only provides a small margin (2-5ns) for clock skew between *CLK* and $\overline{CLK}$. Therefore in the pixel a separate clock buffer is employed to provide this signal. Another matter that requires some consideration is the set-up of data prior to clocking, which needs to be approximately 2ns in lead of the clock signal.

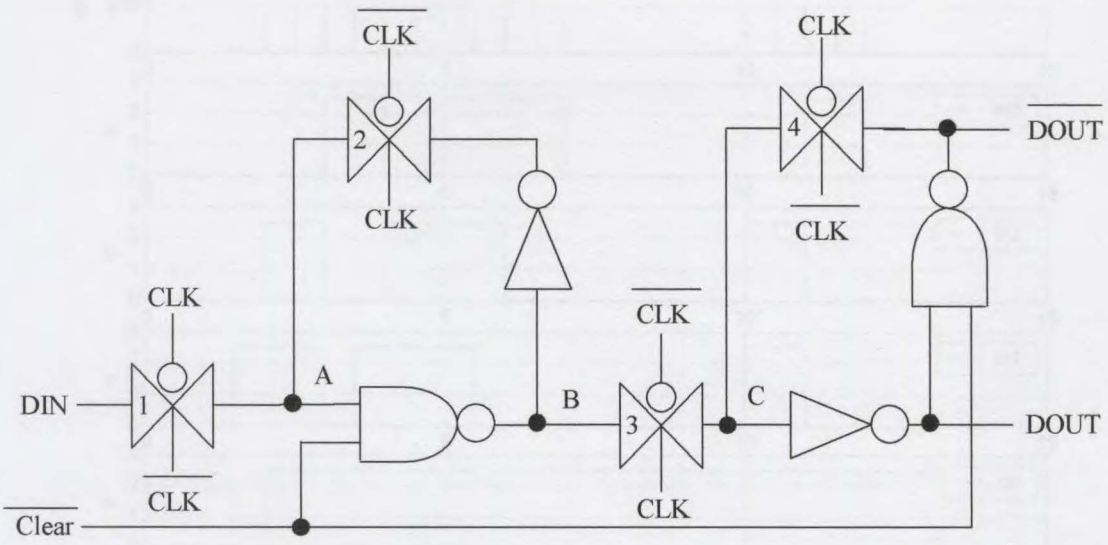


Figure 6-13 DFF Schematic

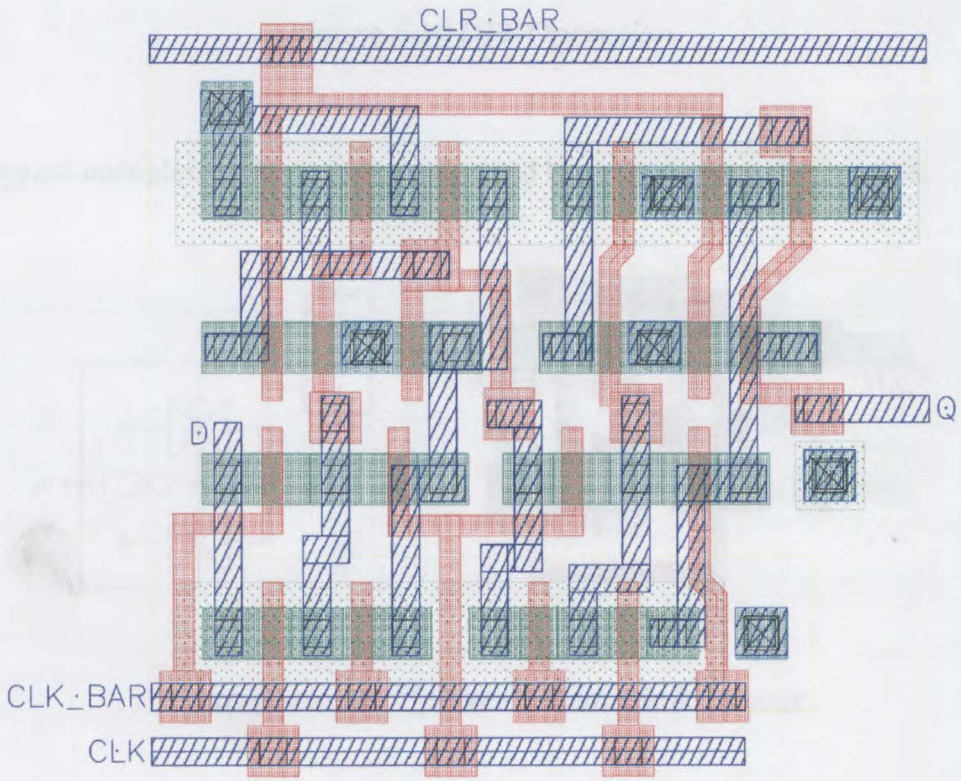


Figure 6-14: DFF Layout

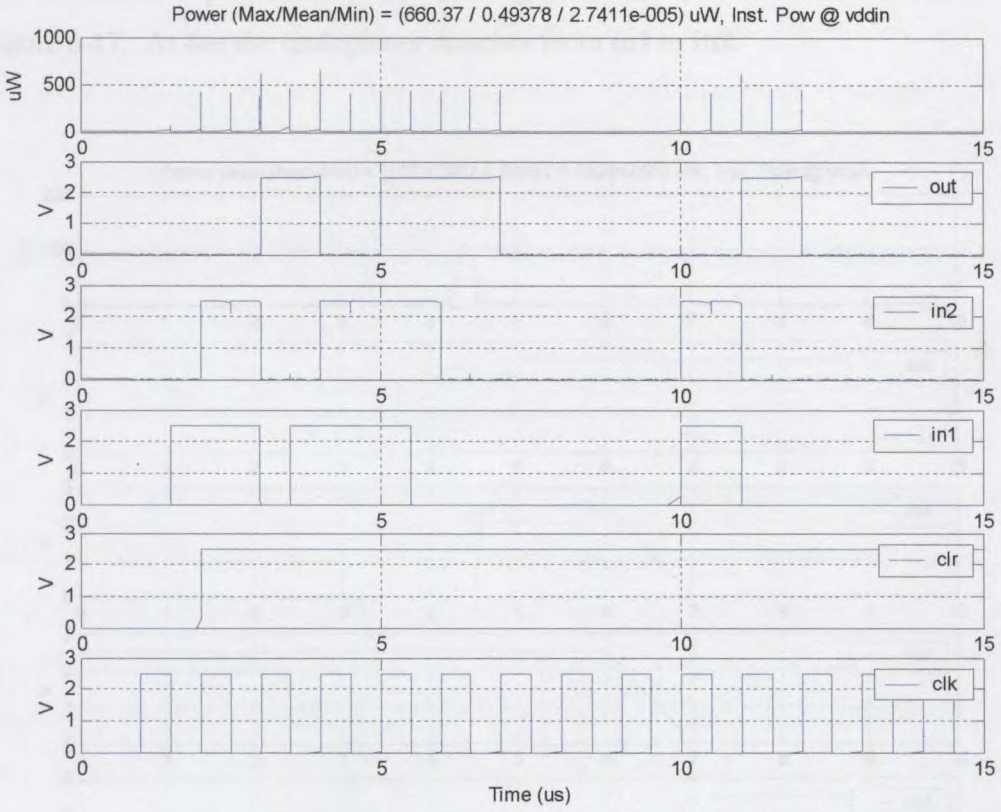


Figure 6-15 DFF Operation

A typical multiplexer schematic and associated layout is shown in Figure 6-16.

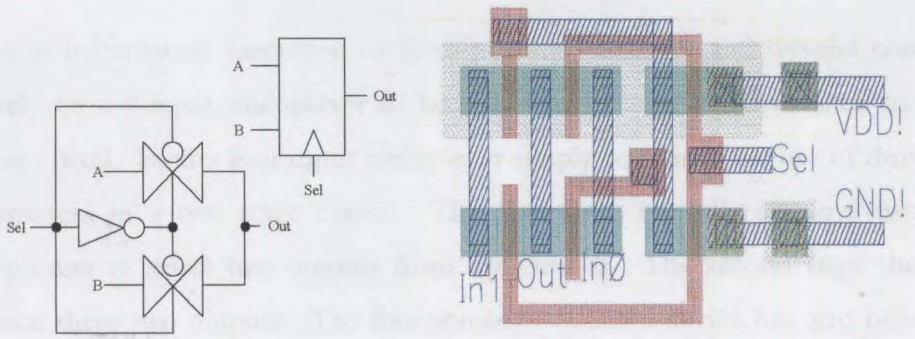


Figure 6-16: Multiplexer Schematic & Layout

The design uses two transmission gates connected inversely to allow either one input **A** or **B** to be switched to the output based on the signal at **Sel**. The layout for this design

fills an area 6.92 μm wide by 4.00 μm high, and its operation can be verified from Figure 6-17. At 4ns the multiplexer switches from **in1** to **in2.**

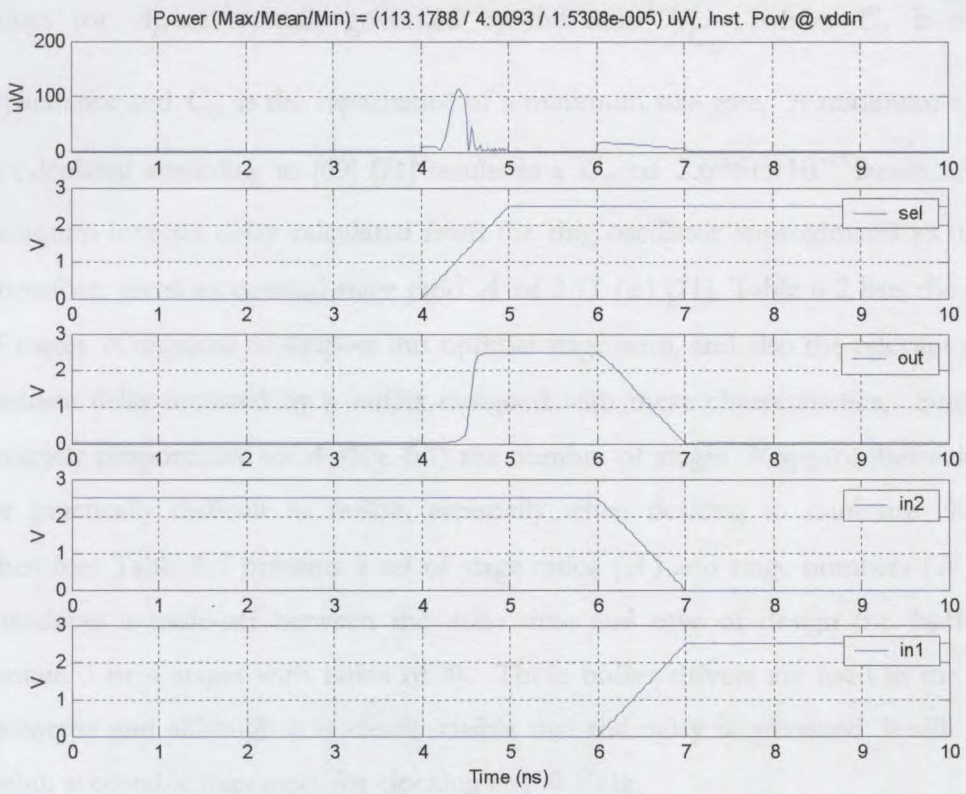


Figure 6-17: Multiplexer Operation

When in information extraction or feed-in mode the ZTE and wavelet components bot rely on a 4-input multiplexer to be switched so as to be connected to the next adjacent pixel. Such a four input multiplexer simply combines the use of three 2-input multiplexers in a two stage circuit. The first stage typically employs two 2-input multiplexers to select two outputs from four inputs. The second stage then selects between these two outputs. The functionality of such a device has also been verified although not presented here.

6.5.2. Buffer Requirements

The IP100P employs several buffers located on the edge of the pixel array to drive the control signal busses in a row based manner. The ZTE component requires the use of

six of these control lines and hence related buffers. Buffer design typically hinges on the selection of suitable values for both the number of stages N and area of the final drive transistor A_D typically in comparison to the area of minimum size gate [71]. The values for A_D are typically governed by the ratio C_L/C_G, where C_L is the load capacitance and C_G is the capacitance of a minimum size gate. A minimum size gate, as calculated according to [69] [71] results in a C_G of 2.6961×10^{-15} farads. Also the minimum inverter delay calculated from the ring oscillator approximates to 0.111 ns. Therefore, given an optimal stage ratio A of 2.71 (e) [71], Table 6-2 lists the number of stages N required to support this optimal stage ratio, and also the relevant minimal intrinsic delay incurred by a buffer designed with these characteristics. Since N is inversely proportional to A (Eq. 6.1) the number of stages N approaches values that are practically difficult to realise, especially when desiring to minimise chip area. Therefore Table 6-3 presents a set of stage ratios (A) and stage numbers (N) which introduces a trade-off between the delay time and ease of design (i.e. buffers that contain 3 or 4 stages with ratios of 4). These buffer drivers are used in the IP100P prototype and although it is clearly visible that the delay is increased, it still remains within acceptable tolerances for clocking at 100 KHz.

Table 6-2: Ideal Stage Ratio of e

			Lumped		C_L	A_D	N		
	Per-Pixel	Pixels	Array Total	Fan	Total Array Load	Drv Cap	Opt Num	Buffer Del	Intrinsic
Signal	Cap (F)	Per Row	Cap (F)	In	Cap (F)	Size Ratio	of Stages	Ratio	Delay (ns)
CLK	9.46E-15	32	3.02628E-13	2	4.75178E-13	176.25	5	14.06	1.562
CFZTR	5.00E-14	32	1.60045E-12	5	2.03182E-12	753.62	7	18.01	2.001
CSYMIO	7.79E-14	32	2.49367E-12	5	2.92504E-12	1084.92	7	19.00	2.111
CLOADR1I	4.02E-14	32	1.28481E-12	4	1.62991E-12	604.54	6	17.41	1.934
HENABLE	2.44E-14	32	7.8232E-13	2	9.54870E-13	354.17	6	15.96	1.773
VENEBLE	2.44E-14	32	7.7968E-13	2	9.52230E-13	353.19	6	15.95	1.772

$$N = \frac{\ln\left(C_L/C_G\right)}{\ln(A)}$$

(Eq. 6.1)

Table 6-3: Selected Stage Ratio

	A_D	N	A			A_{DG}	Drive Cap		D_{tot}
	Drv Cap	Sel Num	Selected	Buffer Delay	Intrinsic	Drive Cap	To Req	Load Delay	Total Delay
Signal	Size Ratio	of Stages	Stage Ratio	Ratio	Delay (ns)	Ratio	Cap Ratio	(ns)	(ns)
CLK	176.25	4	4	16	1.778	256	0.688462501	0.076	1.854
CFZTR	753.62	4	4	16	1.778	256	2.943812359	0.327	2.105
CSYMIO	1084.92	3	4	12	1.333	64	16.95181929	1.883	3.217
CLOADR1R2R4R5	604.54	4	4	16	1.778	256	2.361503653	0.262	2.040
HENABLE	354.17	3	4	12	1.333	64	5.533863729	0.615	1.948
VENEBLE	353.19	3	4	12	1.333	64	5.518563851	0.613	1.946

Therefore, two buffers can be designed, one, a four stage and the other, a three stage (inverting). Figure 6-18 and Figure 6-19 show the layouts for the 3-stage and 4-stage buffers respectively.

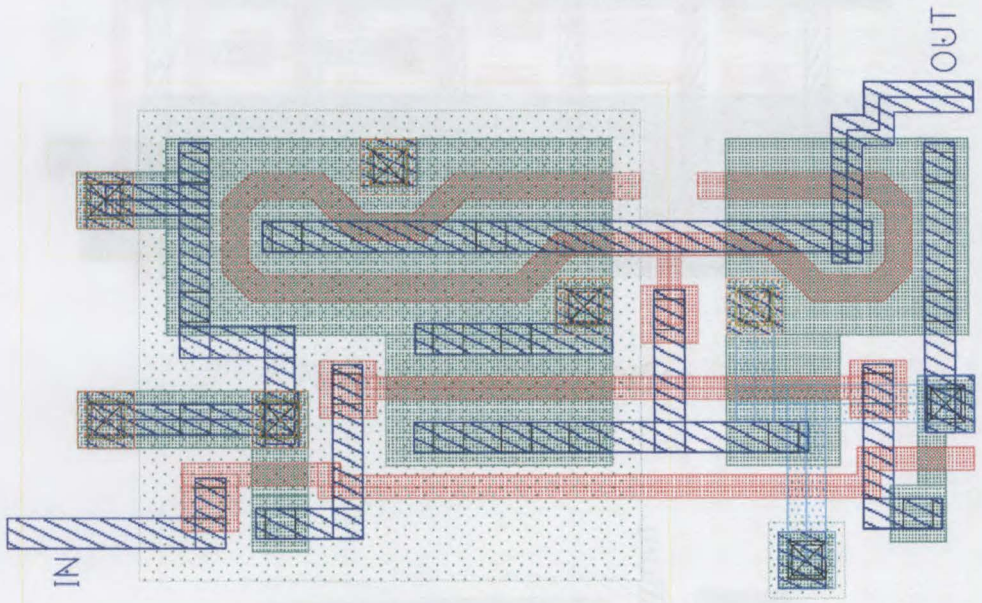


Figure 6-18: 3 Stage Buffer layout

The 3-stage buffer is 10.09 x 5.520μm in size while the 4-stage buffer is 9.98 x 12.7μm for the height and width respectively.

Since each buffer drives a row of 32 pixels and since there are 32 rows, a further buffering stage is required to drive this stage from the pad signal. In the case of the three stage buffer, another 3-stage buffer is used to drive the 32 secondary buffers,

therefore providing a non-inverting signal to the pixels. The rows with 4-stage buffers, on the other hand, are driven by a third 2-stage buffer. This 2-stage buffer, layout in Figure 6-20, has an intrinsic delay of $(2)(4)(t_d)$ = 0.889ns and has a total delay of 0.889(2) =1.778ns. It is 9.91μm high and 3.750μm wide.

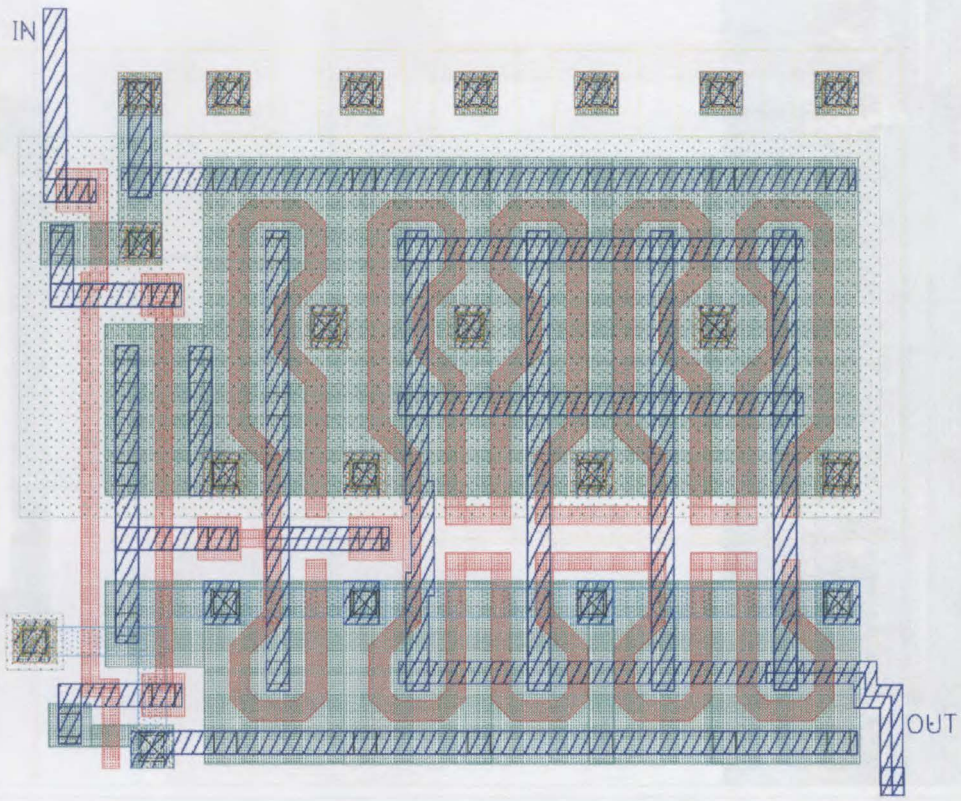


Figure 6-19: 4-Stage Buffer

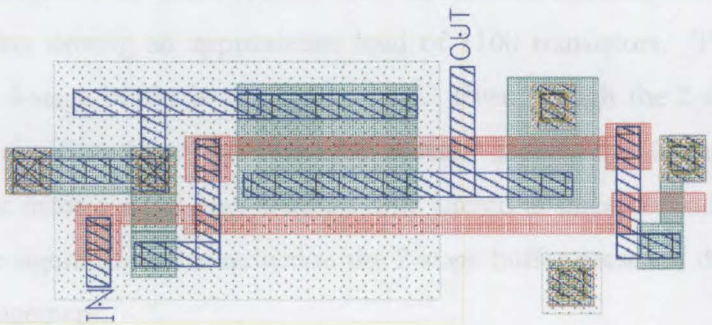


Figure 6-20: 2-Stage Buffer

Figure 6-21 illustrates the technique implemented when connecting the pad to the internal buffers. Here a 2-stage buffer is connected via a bus to 32 4-stage buffers for the control signal CLOADR1R2R4R5.

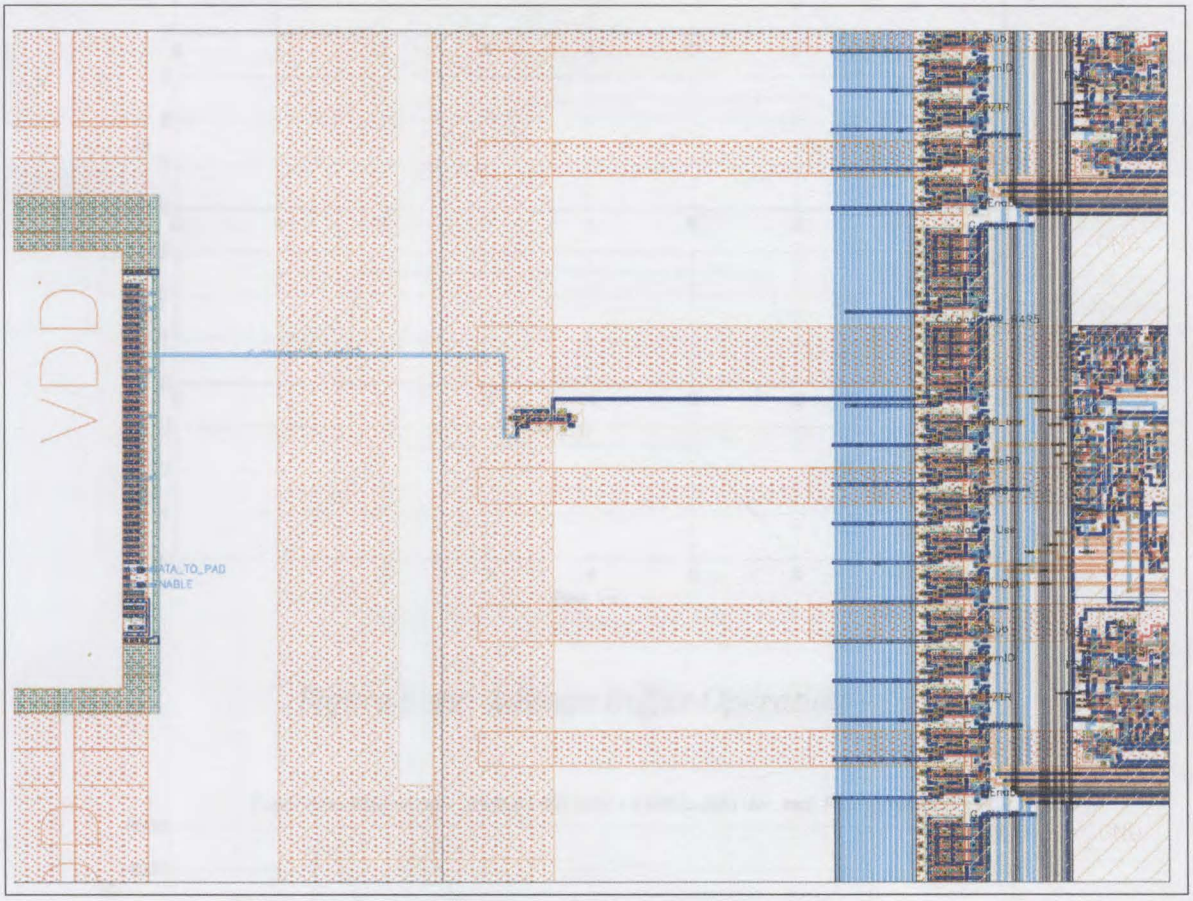


Figure 6-21: Pad to Buffer Connections

Figure 6-22 and Figure 6-23 show results from simulations conducted on both the 4 and 3 stage buffers driving an approximate load of 1100 transistors. The "cleaner" operation of the 4-stage buffer is clearly observed. Even though the 2-stage buffer's performance is significantly worse than the 4-stage buffer, the 2-stage buffer is sufficient to drive most control lines that are only altered at microsecond boundaries. Hence due to the significance size reduction the 2-stage buffer presents, this allows for better space management.

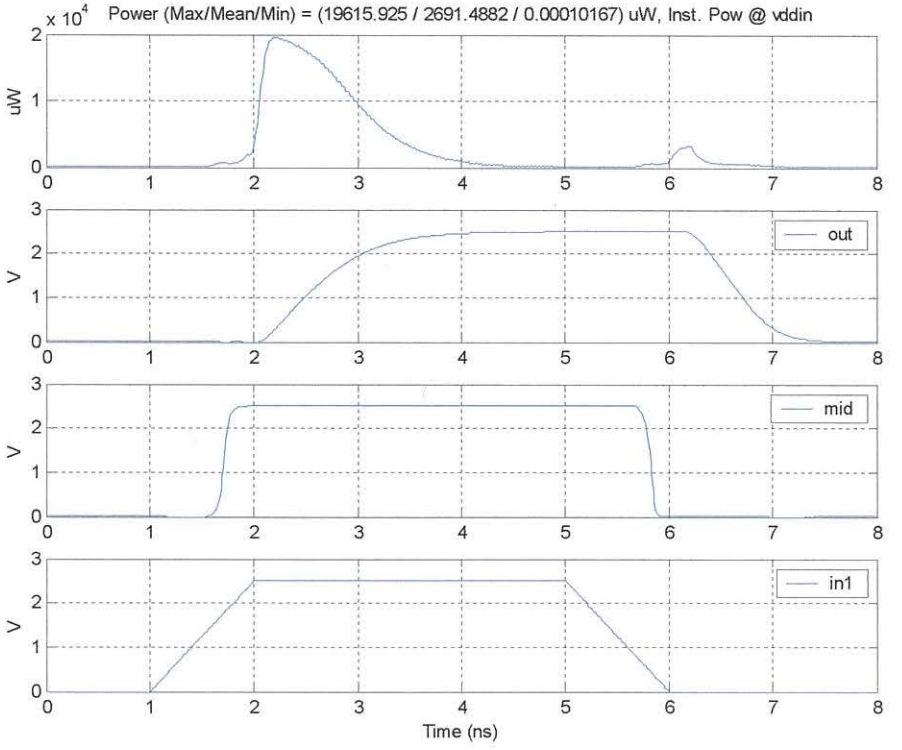


Figure 6-22: 4-Stage Buffer Operation

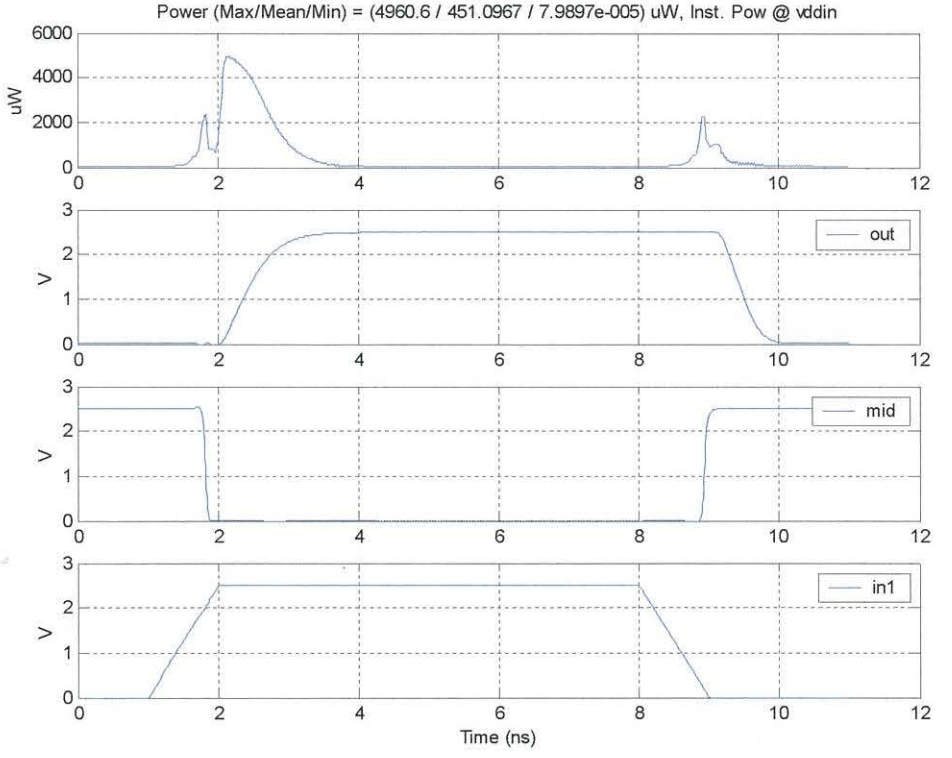


Figure 6-23:3-Stage Buffer Operation

6.6. ZTE Post Layout Simulations

This section details the simulations performed to verify the correct operational functionality of the ZTE component in the IP100P. Due to the difficulties associated with performing simulations on the entire array (~ 900,000+ transistors) without the use of appropriate tools such as Star-Hspice, the operational functionality of a single pixel is simulated. The simulations are organised into two sections, the ZID part which simulates an eight bit register combined with a significance identification module, and a ZTE part which performs all of the other ZTE symbol generation and extraction process. Control and status signals attached to each of these components are presented in Table 6-4 and Table 6-5. The layout for the ZTE is shown in Figure 6-24. It comprises of 170 transistors and occupies and area of 32x22μm + 18x9μm = 866μm^2. The area without circuitry is reserved for routing via Metal-2 (Metal-2 Free Zone)

Table 6-4: ZID Module Signals

Signal	Direction	Description
Ina	IN	Register LSB Value
Inb	IN	Register Bit1 Value
GENABLE	IN	Clear Signal
ZEROID (out)	OUT	Zero NOT Detected Signal
MID	NA	Signal From XOR
Rin	IN	Data in Register R0

Table 6-5: ZTE Module Signals

Signal	Direction	Description
CLK	IN	General Clock In Signal
CSYMIO	IN	Generate ZTE Symbols Control Signal
CFZTR	IN	Form Zerotree Control Signal
CLR1R2R4R5	IN	Shift & Load Registers Control Signal
HENABLE	IN	Horizontal Pixel Enable Signal
VENABLE	IN	Vertical Pixel Enable Signal
ZEROID	IN	Zero NOT Detected Signal
DATAIN	IN	Incoming Data From Adjacent Pixel
WCDATAOUT	IN	Incoming Wavelet Coefficient
GENABLE	OUT	Pixel Disabled Signal & Clear
DATAOUT	OUT	Data For Next Adjacent Pixel
SSIN	IN	Sibling Significance In Signal
CSIN	IN	Child Significance In Signal
PSIN	IN	Parent Significance In Signal
SOUT	OUT	Sibling Significance Out Signal
PSOUT	OUT	Parent Significance Out Signal

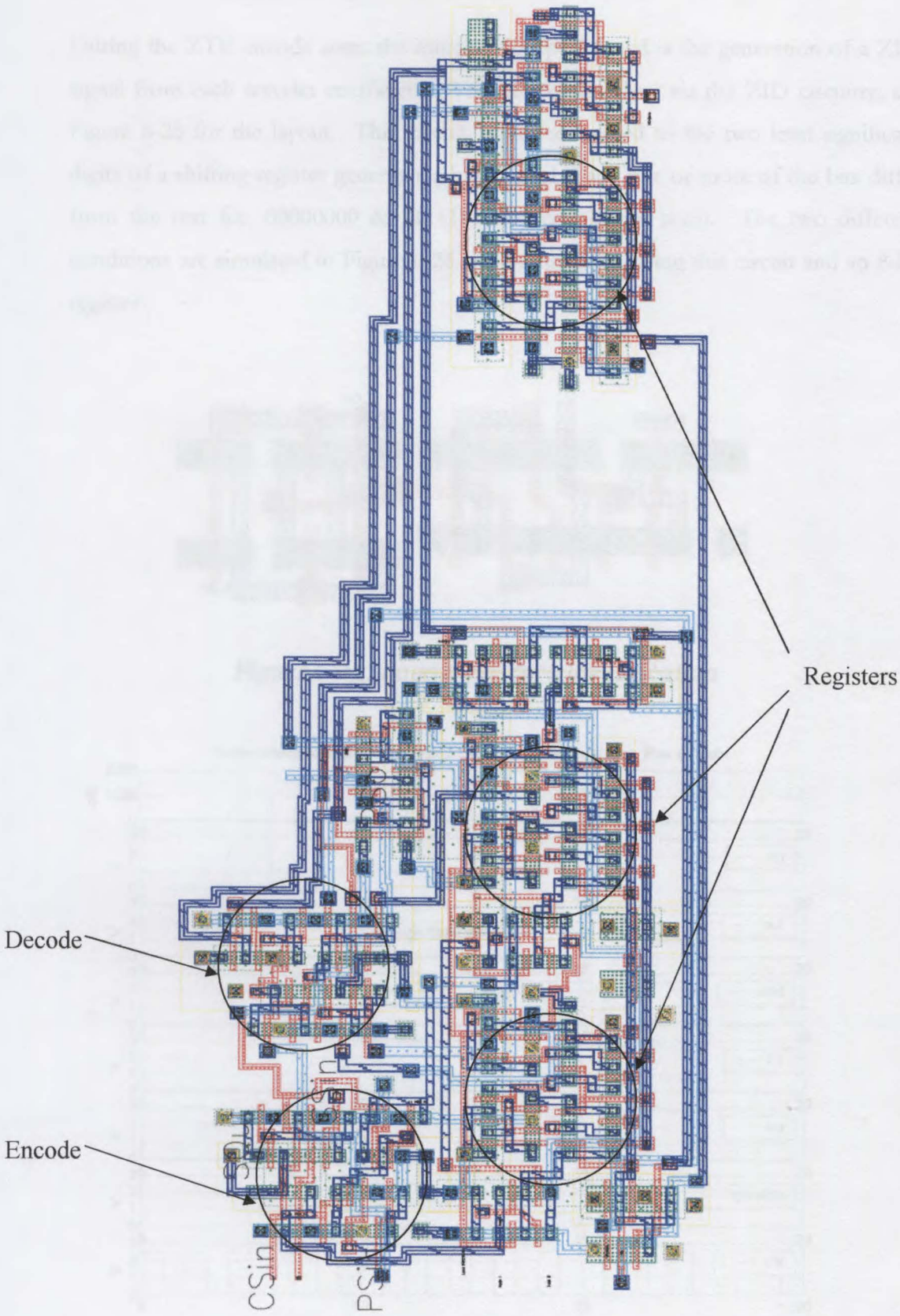


Figure 6-24: ZTE Coding Full Layout Circuitry

6.6.1. Coefficient Significance detection

During the ZTE encode stage the initial action performed is the generation of a ZID signal from each wavelet coefficient. This is accomplished via the ZID circuitry; see Figure 6-25 for the layout. This circuit, when connected to the two least significant digits of a shifting register generates a high signal if any one or more of the bits differ from the rest (i.e. 00000000 & 11111111.are considered zero). The two different conditions are simulated in Figure 6-26 and Figure 6-27, using this circuit and an 8-bit register.

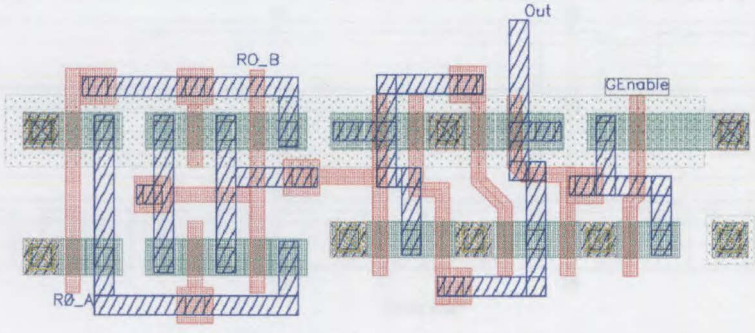


Figure 6-25: Layout for Zero Identification

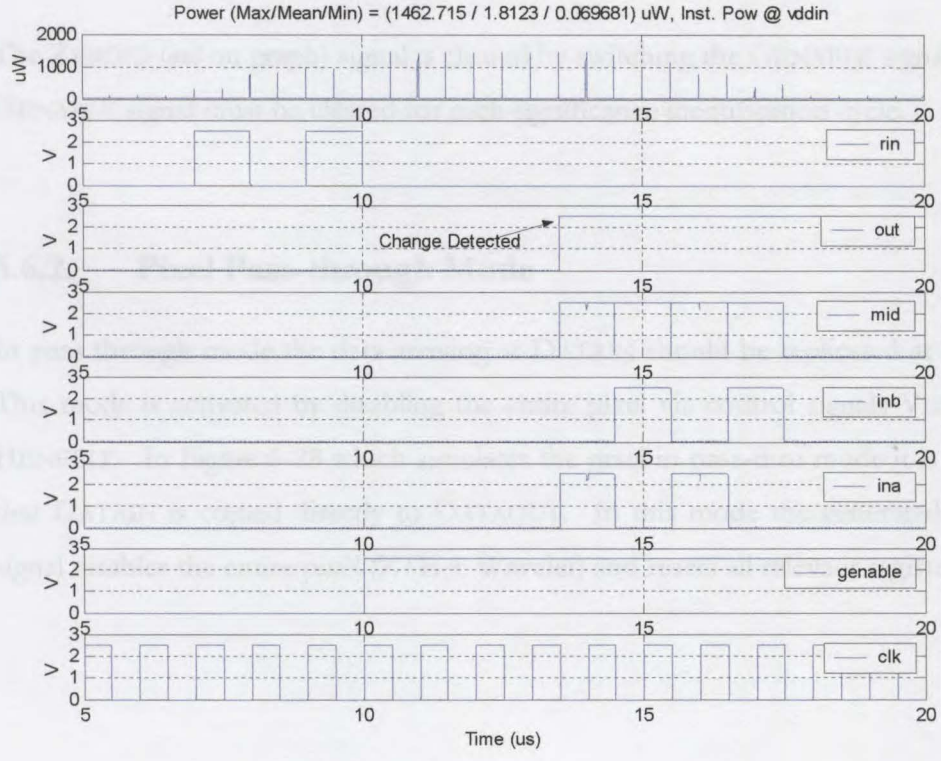


Figure 6-26: ZID Non Zero Detection

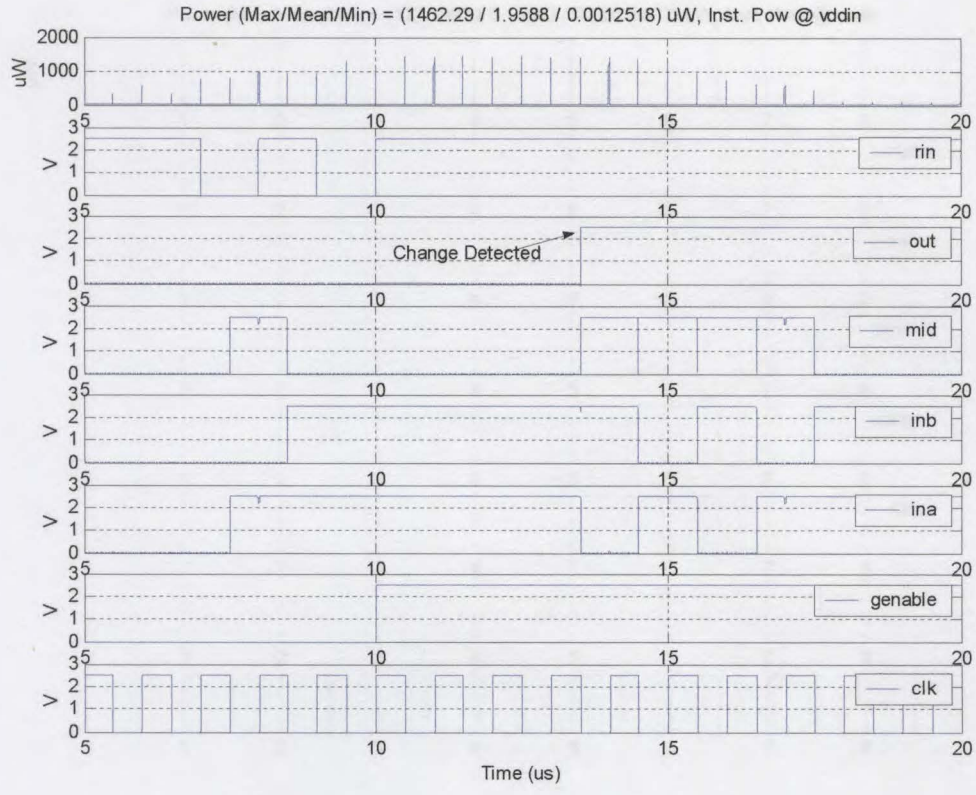


Figure 6-27: ZID Non High Detection

The ZEROID (*out* on graph) signal is cleared by switching the GENABLE signal low. The GENABLE signal must be cleared for each significance identification cycle.

6.6.2. Pixel Pass-through Mode

In pass through mode the data arriving at DATAIN should be replicated at DATAOUT. This mode is activated by disabling the entire pixel via control signals VENEBLE and HENEBLE. In Figure 6-28 which simulates the pixel in pass-thru mode it is easily seen that DATAIN is copied directly to DATAOUT. In this mode the generated GENABLE signal disables the entire pixel (ZTE + Wavelet) and resets all relevant registers.

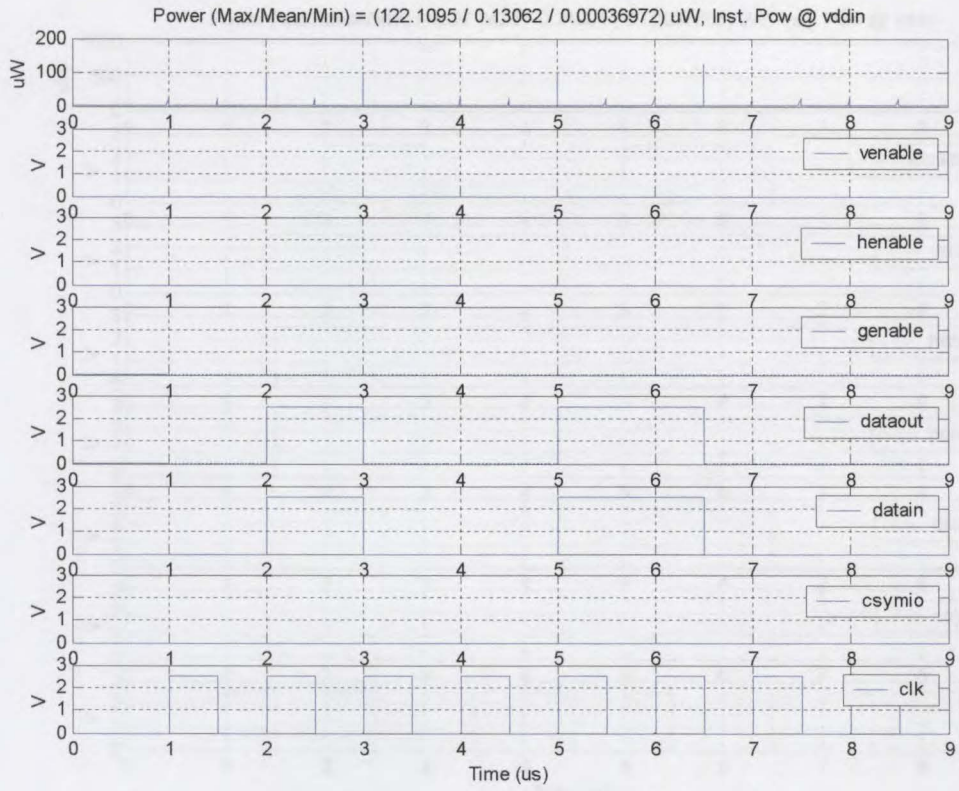


Figure 6-28: Pixel Pass-thru Mode

6.6.3. Wavelet Coefficient Output

This mode is used to extract coefficients from the wavelet component to a neighbouring pixel. When the control signal CSYMIO is held low and the pixel is activated via GENEBLE, the contents arriving at WCDATAOUT are routed to DATAOUT. Figure 6-29 shows this process, and also highlights the activation of GENABLE only when both HENABLE and VENABLE are set high.

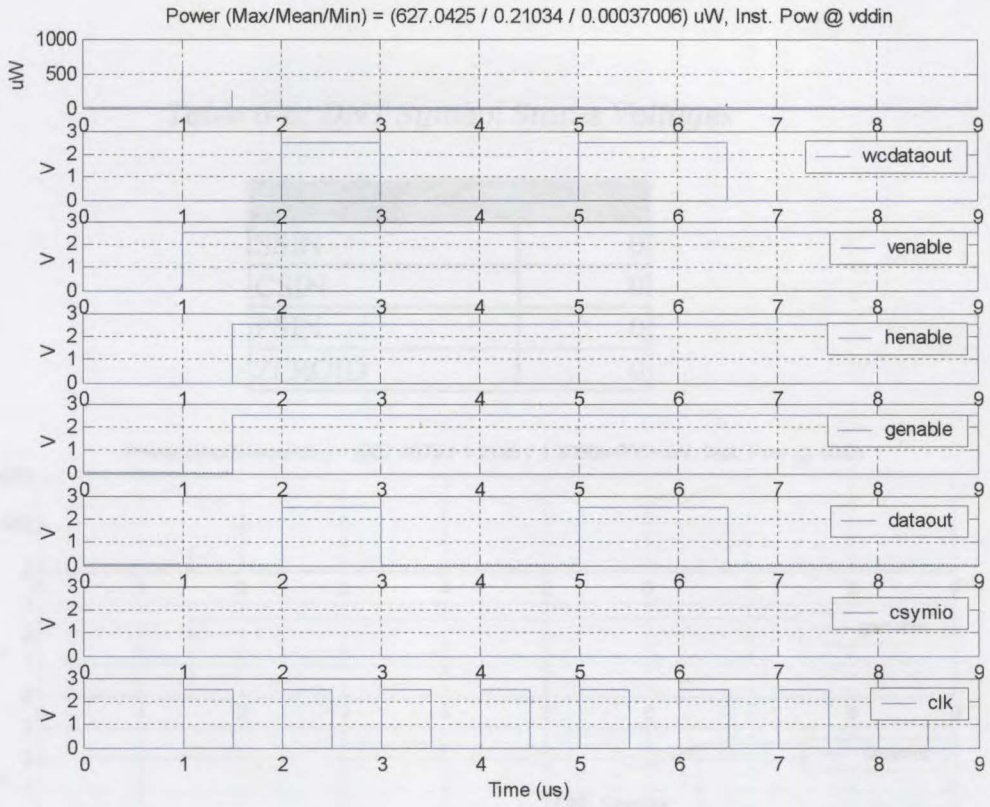


Figure 6-29: WCDATAOUT Mode

6.6.4. DNT Symbol Generation

As with all symbol generation the DNT symbol is based on a two-part process (Symbols & Coefficients) (Section 5.4.3). This two-part process is verified in the following section for each operation mode. Since the decode mode and the coefficient encode part follow a similar control timing scheme they are grouped together in the symbol decode section following the current symbol generation sections. During the encode process when the ZTE status signals assume the values in Table 6-6 a two bit Do Not Transmit (DNT) symbol (2.5, 2.5 V) is generated and latched into R4 and R5. The timing and simulation for this state is presented in Figure 6-30. The signal LR1R2R4R5 initially routes the symbols generated from the ZTE status signals to R4 and R5, which latches this data when the clock enabling signal CSYMIO is set high. When CSYMIO becomes active again the symbols are shifted out while receiving symbols from adjacent pixel from DATAIN. This procedure is applied for the generation of all other symbols.

Table 6-6: DNT Symbol Status Voltages

Signal	Value (V)
SSIN	0
CSIN	0
PSIN	0
ZEROID	0

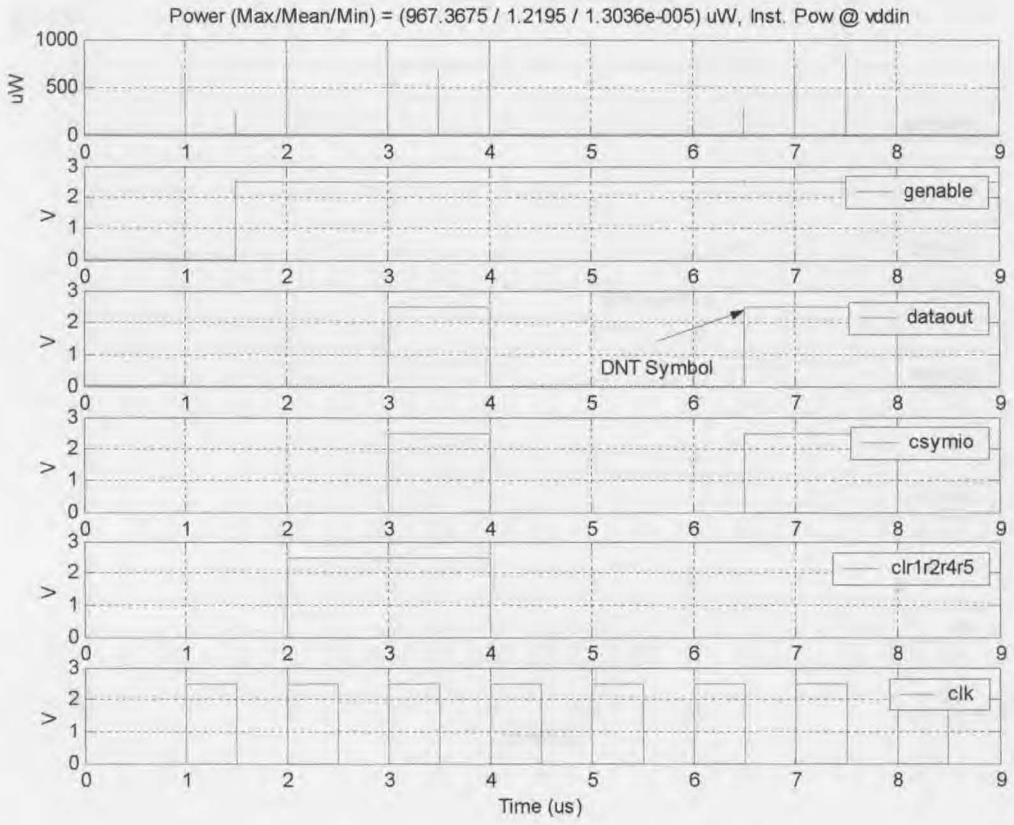


Figure 6-30: DNT Detection, Latch & Shift Out

6.6.5. ZTR Symbol Generation

The ZeroTree Root or (ZTR) symbol, signal level (0, 2.5V), results when a ZTE status signal corresponding to that of Table 6-7 is detected. The timing and simulation for the generation, latching and extraction are presented in Figure 6-31.

Table 6-7: ZTR Symbol status Voltages

Signal	Value (V)
SSIN	0 or 2.5
CSIN	0
PSIN	2.5
ZEROID	0

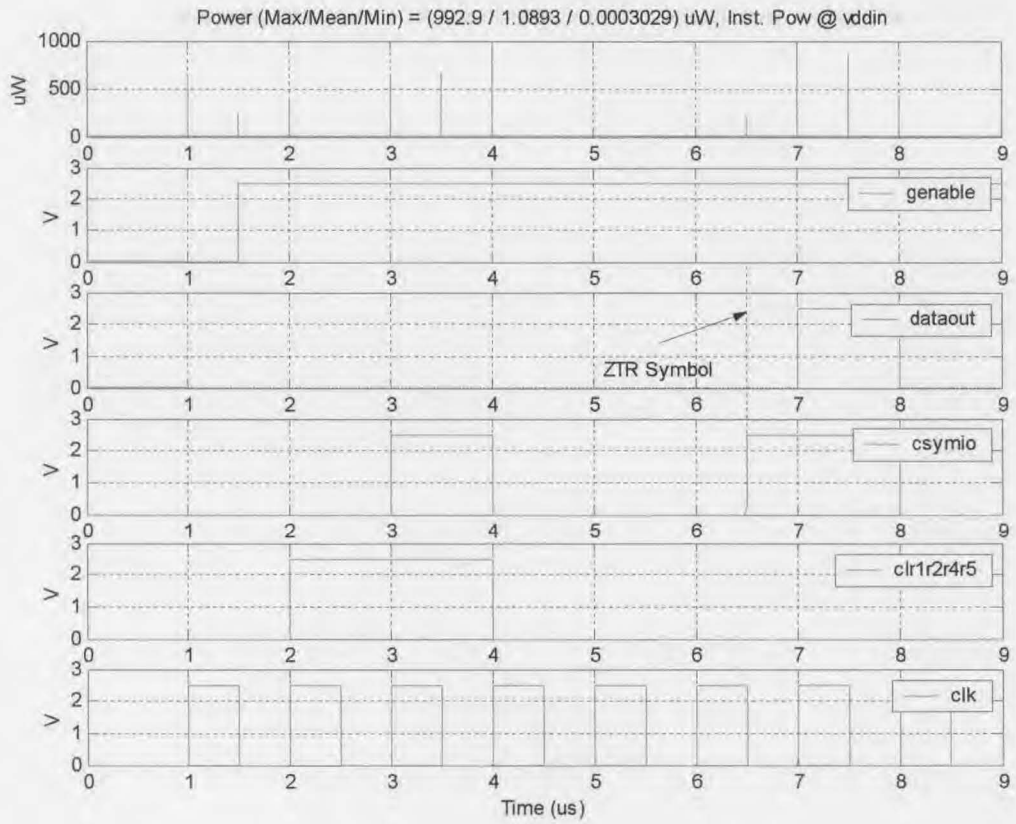


Figure 6-31: ZTR Detection, Latch & Shift Out

6.6.6. VZT Symbol Generation

The Valued Zero Tree root (VZT) symbol, with signal level (0, 2.5V) is generated when the ZTE status signals resemble that of Table 6-8. The timing and simulation for the detection, latch and extraction is seen in

Table 6-8: VZT Status Signals

Signal	Value (V)
SSIN	0 or 2.5
CSIN	0
PSIN	2.5
ZEROID	2.5

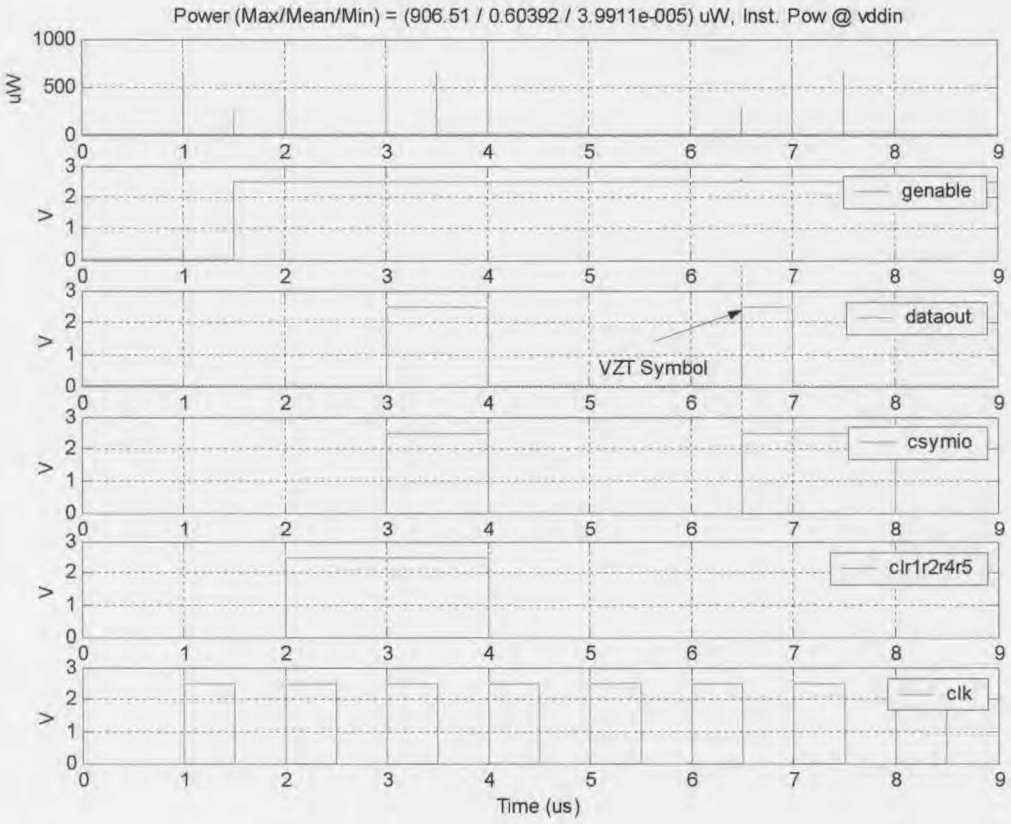


Figure 6-32: VZT Detection, Latch & Shift Out

6.6.7. VAL Symbol Generation

This symbol, the VALue with signal levels (0, 0V), is generated when the ZTE status signals equate to that in Table 2-1. Timing and simulation results for the generation of this symbol is presented in Figure 6-33.

Table 6-9: VAL Status Signals

Signal	Value (V)
SSIN	0 or 2.5
CSIN	2.5
PSIN	2.5
ZEROID	0 or 2.5

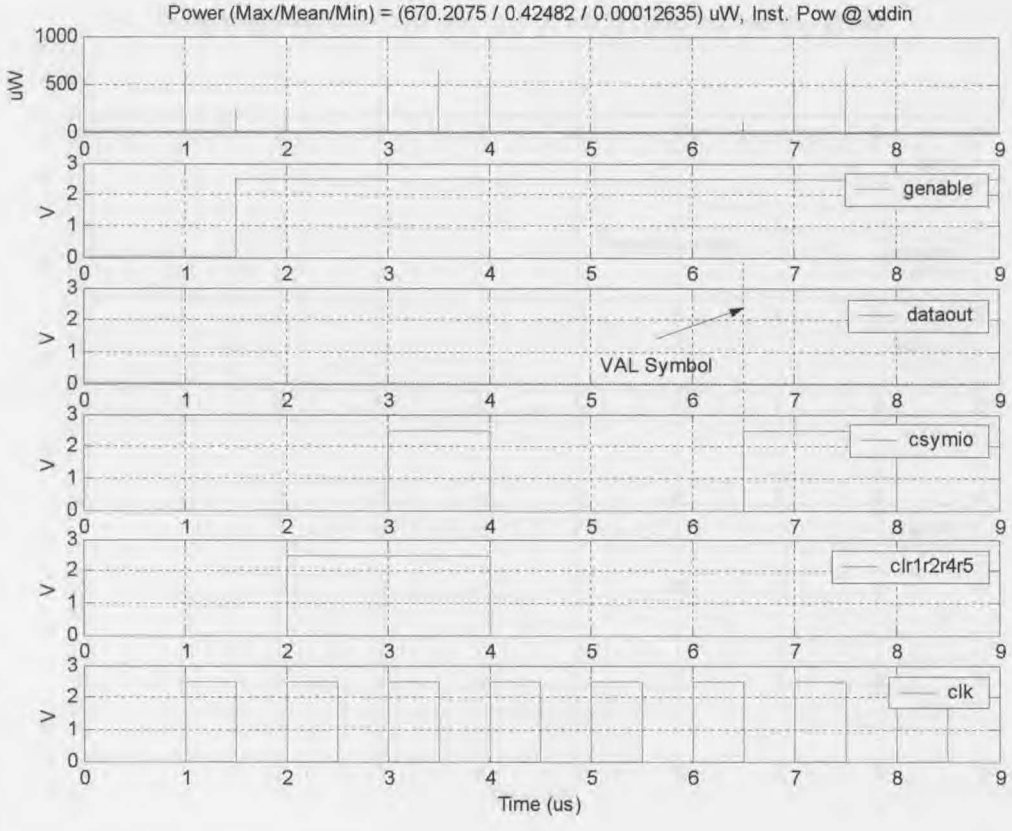


Figure 6-33: VAL Detection, Latch & Shift Out

6.6.8. DNT Decode Operation

In the decode phase a DNT valued pixel is identified by its immediate parent. Here if the PSIN signal is at 2.5V, then the pixel is disabled and it enters pass-thru mode, which then never expects a symbol or a wavelet coefficient. In encode mode one the DNT symbol has been extracted the pixel enters this pass-thru mode to avoid transmitting the wavelet coefficient. The control signalling for this mode of operation requires that

initially, all pixels be set to the symbol VAL (0V, 0V), which is achieved by disabling the GENABLE signal, via HENABLE and VENABLE. Following this if the CFZTR signal is held high while the PSIN signal remains high then the pixel will enter pass-thru mode. The operational simulation corresponding to this mode is displayed in Figure 6-34. The DNT decode mode is unique in that it solely relies on its parent to supply activation information, hence hot requiring a symbol itself.

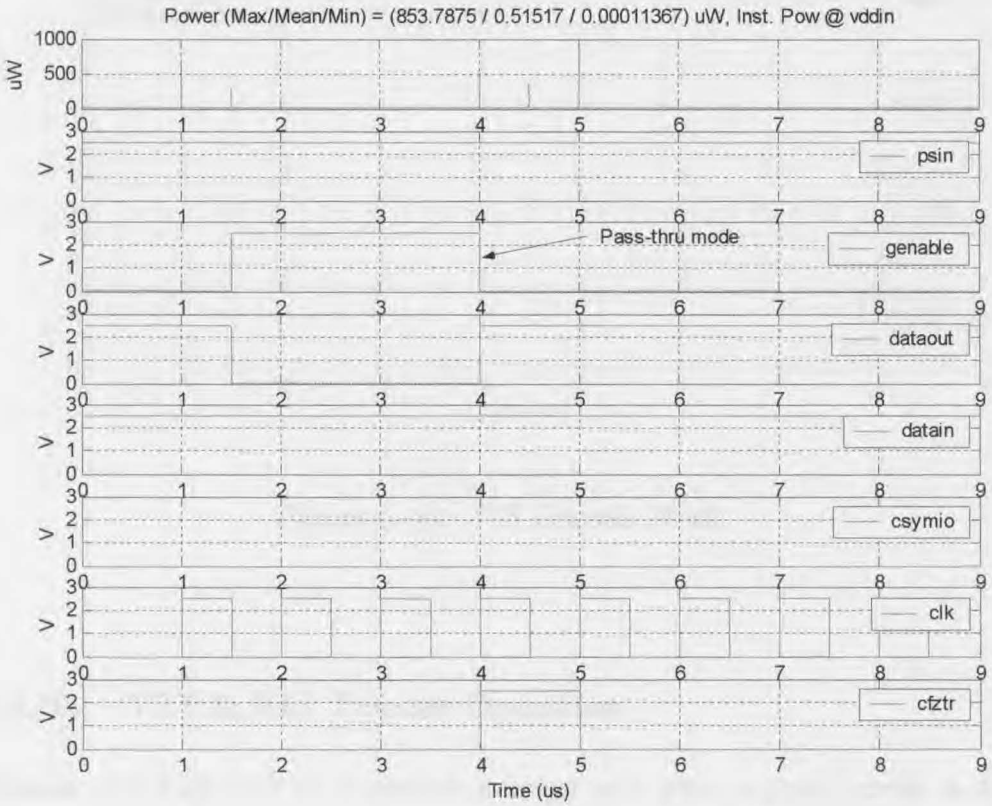


Figure 6-34: DNT Decode Mode

6.6.9. ZTR Decode Operation

If a pixel's parent is significant (PSIN = 0V) then it should expect a symbol. In the case where this symbol is a ZTR (0V, 2.5V) it implies that a wavelet coefficient does not follow, hence the pixel deactivates itself (GENABLE is set low). The operation of this mode is presented in Figure 6-35.

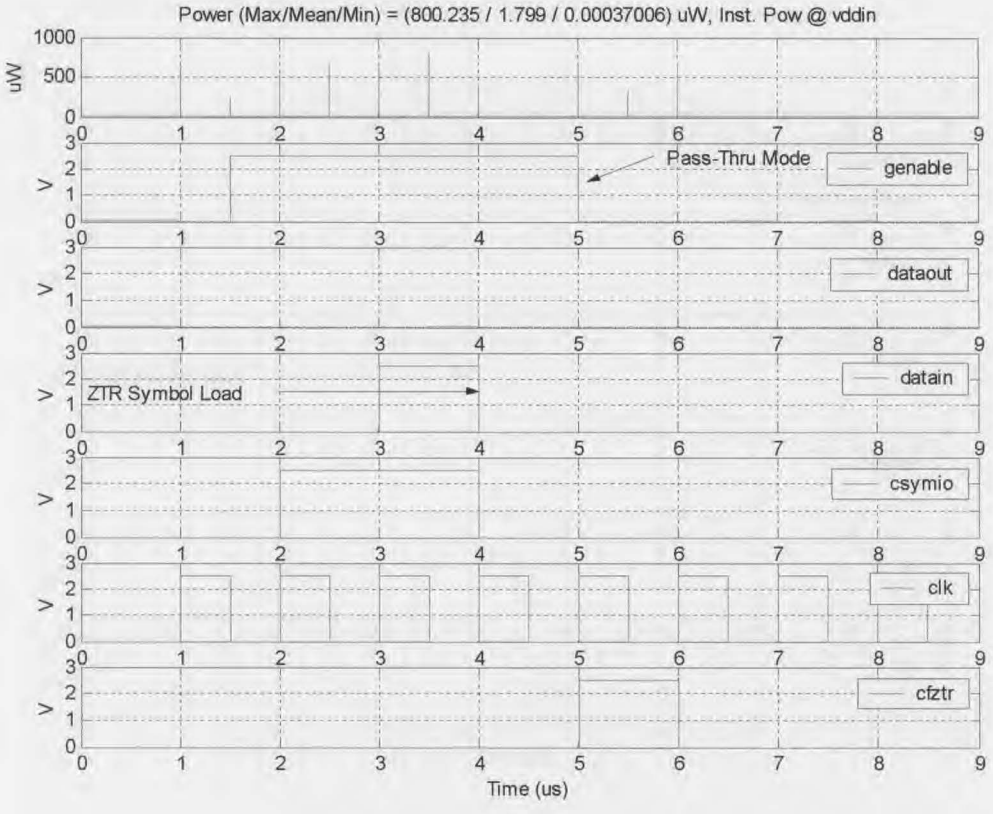


Figure 6-35: ZTR Decode Mode

6.6.10. VZT & VAL Decode Operation

Decode of a VZT or VAL symbol is activated only when a pixel's parent is deemed significant (PSIN = 0V). Both VZT and VAL decode require a symbol and coefficient to be captured from the stream, therefore the GENABLE signal must remain high after the symbols are captured. When GENABLE is returns to active, any coefficients passed to this pixel will propagate to the wavelet component. Figure 6-36 and Figure 6-37 illustrate the decode operation for a VZT symbol and a VAL symbol respectively.

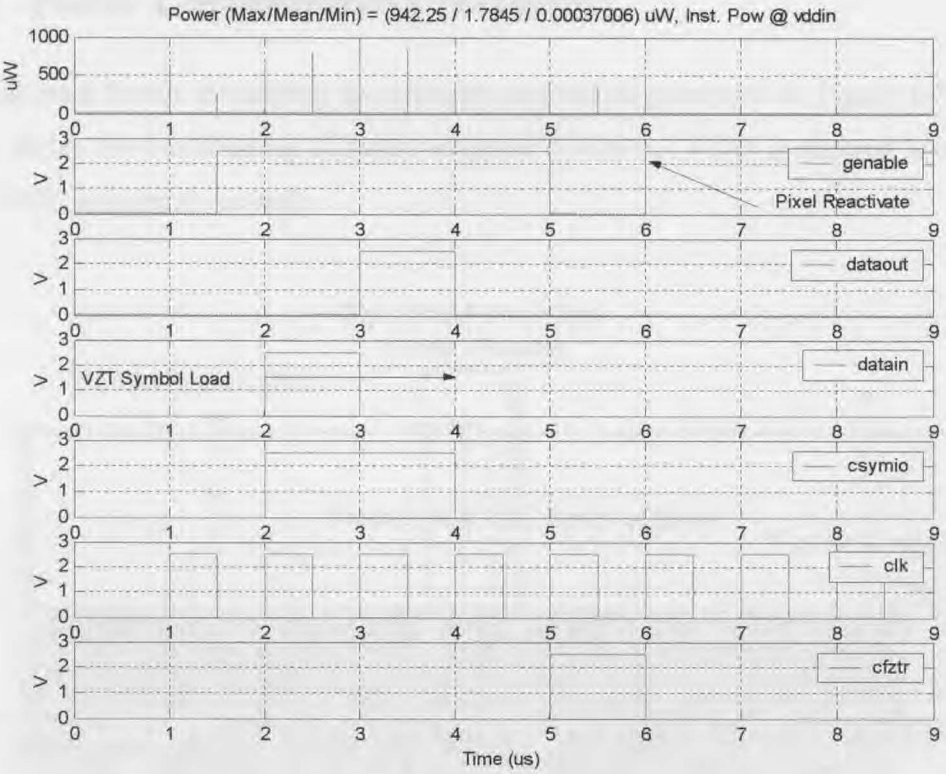


Figure 6-36: VZT Decode

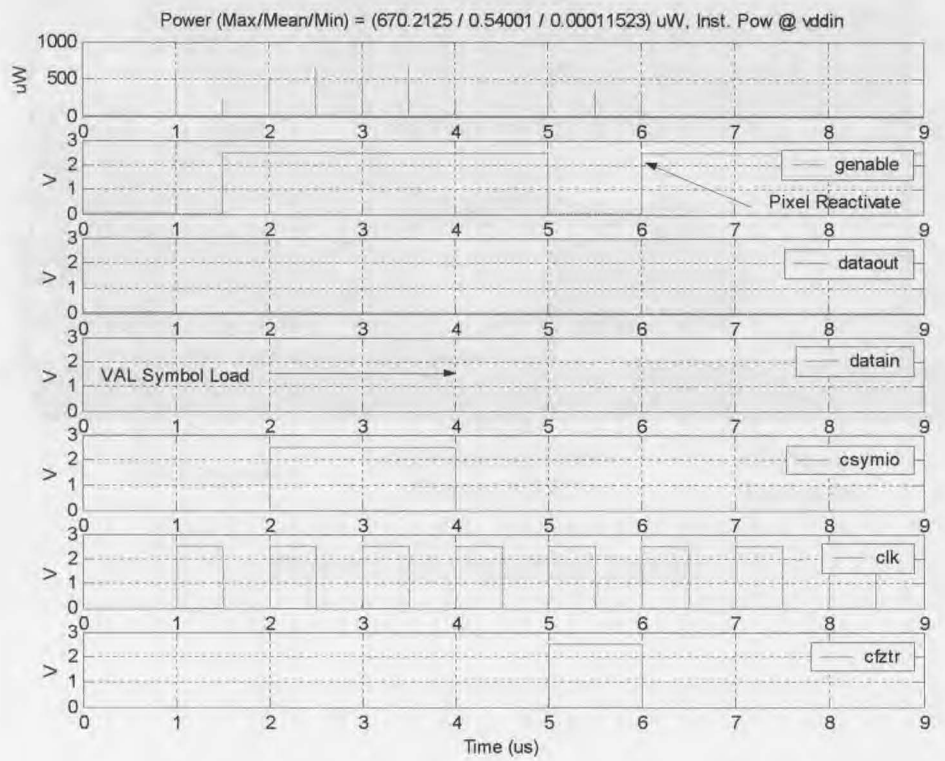


Figure 6-37: VAL Decode

6.7. Pixel Components (Layout)

The full pixel layout identifying the relevant sections is presented in Figure 6-38. This layout allows for the abutting of pixels in either direction. VDD is abutted horizontally while GND is abutted vertically.

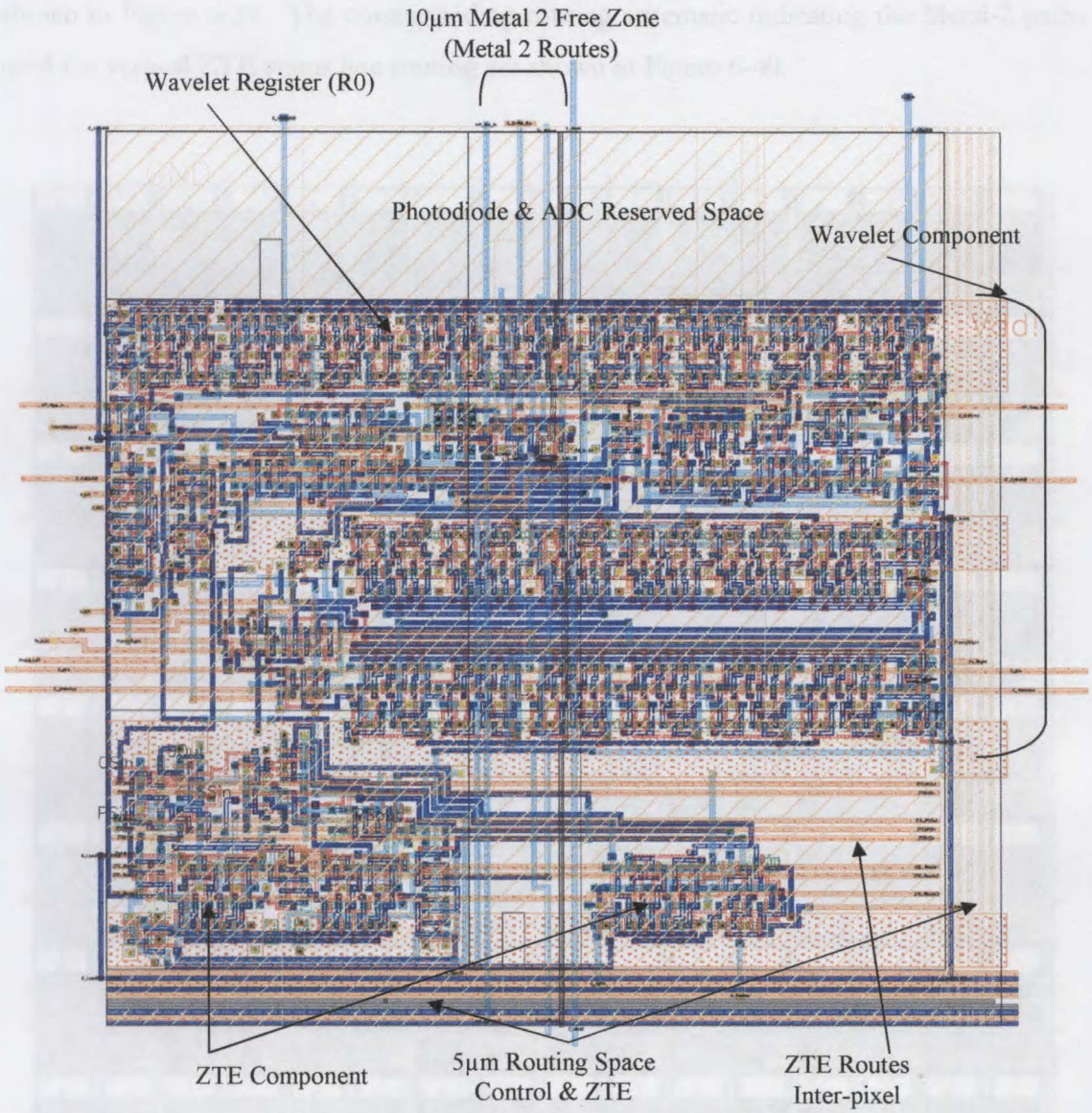


Figure 6-38: Full Pixel Layout

6.8. NBLOCK Layout

The NBLOCK is the smallest replicable component within the IP Array, which is capable of performing a 3-scale image encode / decode in this case. The inter-pixel ZTE status signal routing for each pixel within the NBLOCK is unique, yet identical to the same pixel in another NBLOCK. The full layout constructed with abutting pixels for the NBLOCK is shown in Figure 6-39. The corresponding routing schematic indicating the Metal-2 paths used for vertical ZTE status line routing are shown in Figure 6-40.

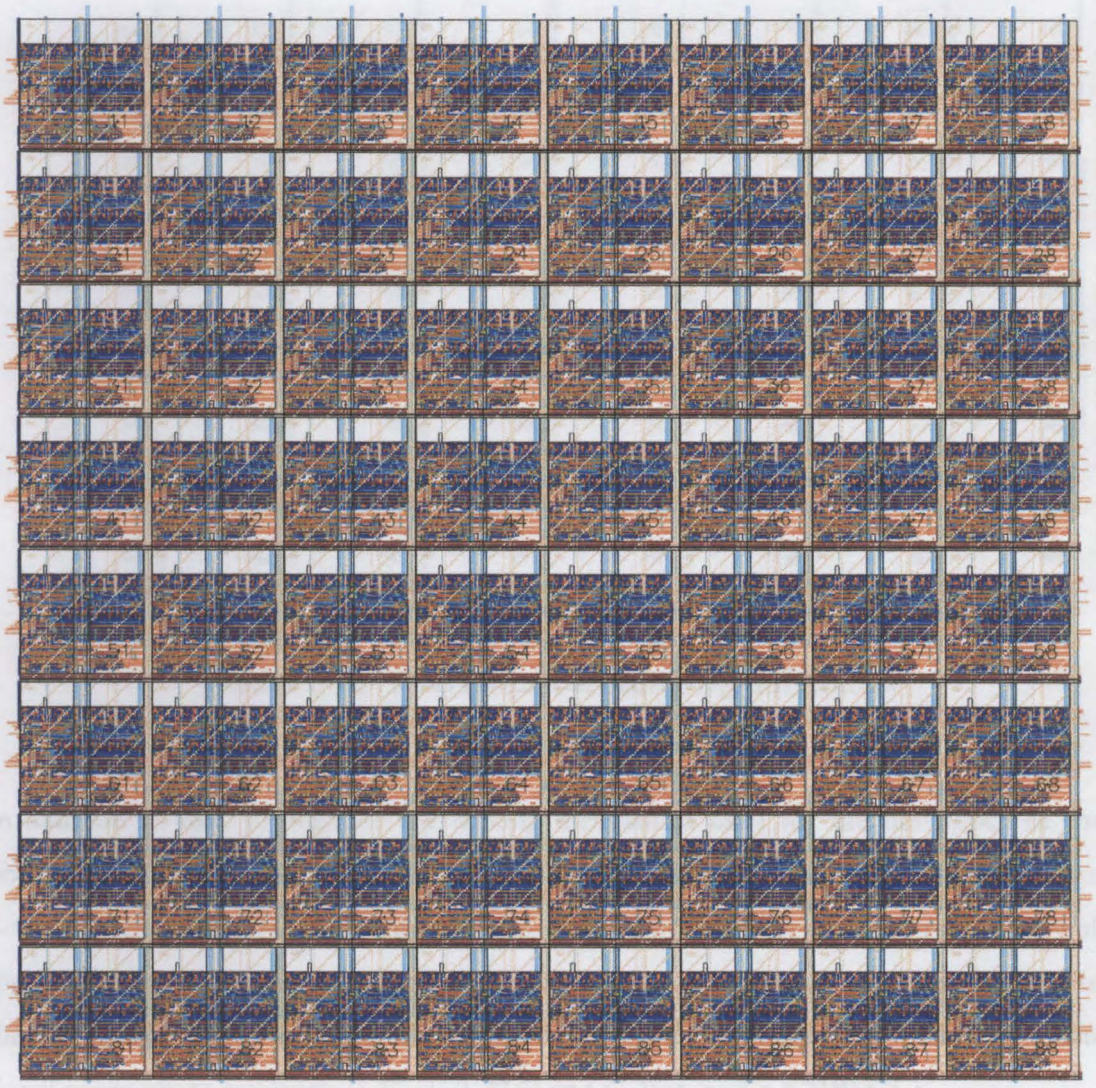


Figure 6-39: 8x8 NBLOCK Layout

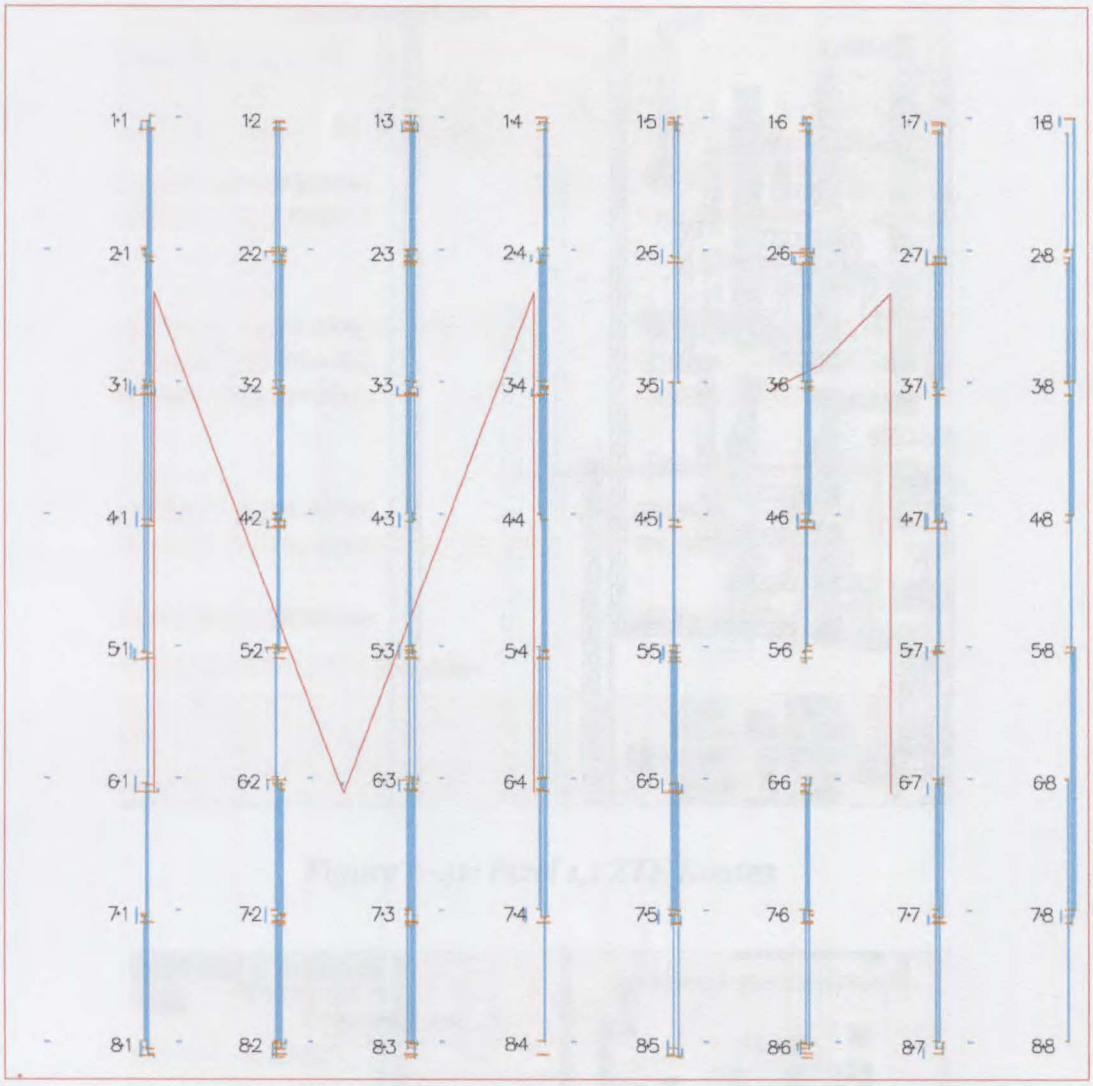


Figure 6-40: ZTE NBLOCK Status Signal Routes

In addition to this, to illustrate the routing mechanism used for the first block of four pixels in an NBLOCK, the four figures Figure 6-41, Figure 6-42, Figure 6-43 and Figure 6-44 are presented. The blue (Metal 2) lines correspond to the vertical routing paths, while the brown (Metal 3) lines correspond to the horizontal route paths. Connections between the two are by means of vias. These figures illustrate the unique pixel-wise ZTE status signal connections for pixels (1,1), (1,2), (2,1) and (2,2) in an 8 x 8 NBLOCK.

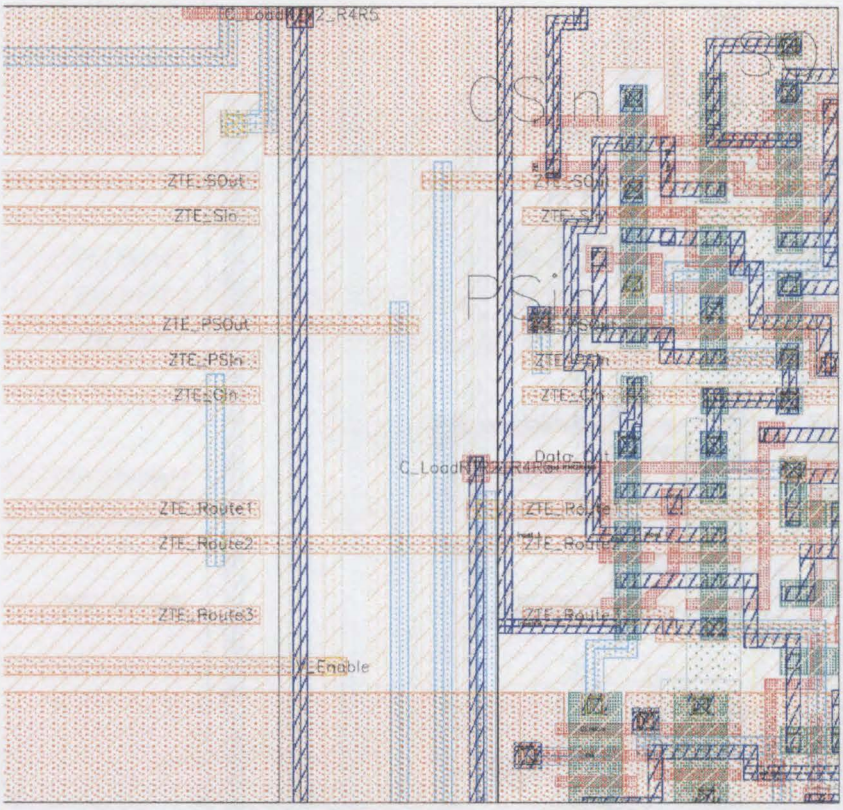


Figure 6-41: Pixel 1,1 ZTE Routes

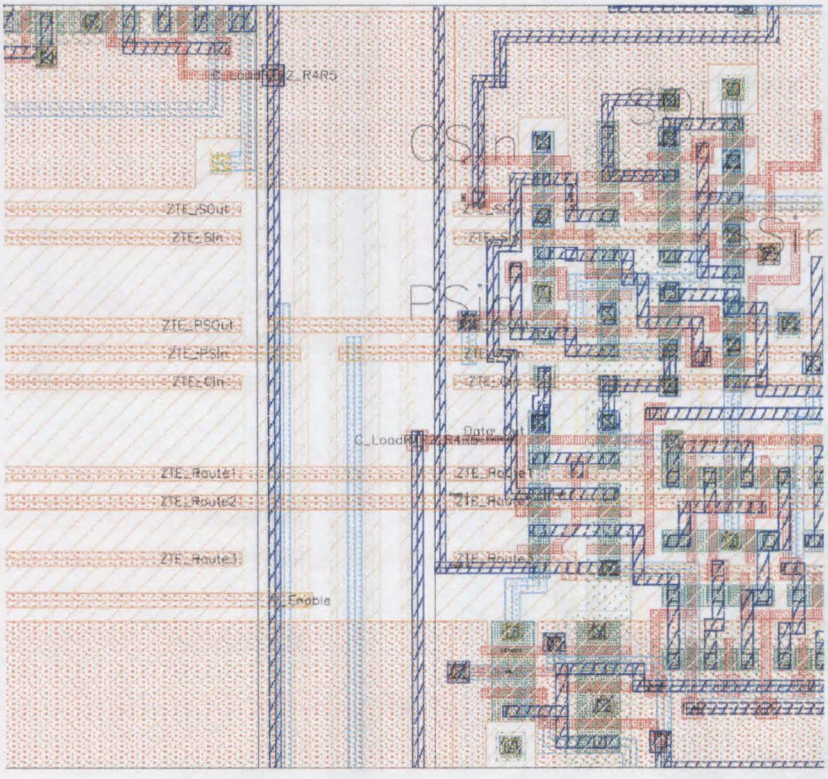


Figure 6-42: Pixel 1,2 ZTE Routes

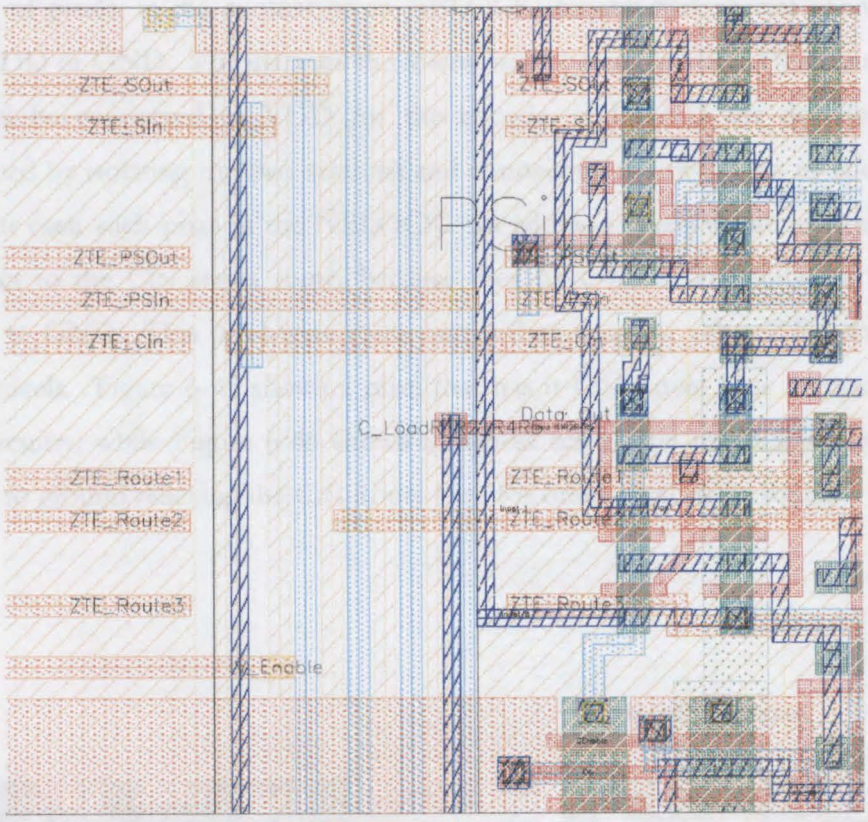


Figure 6-43: Pixel 2,1 ZTE Routes

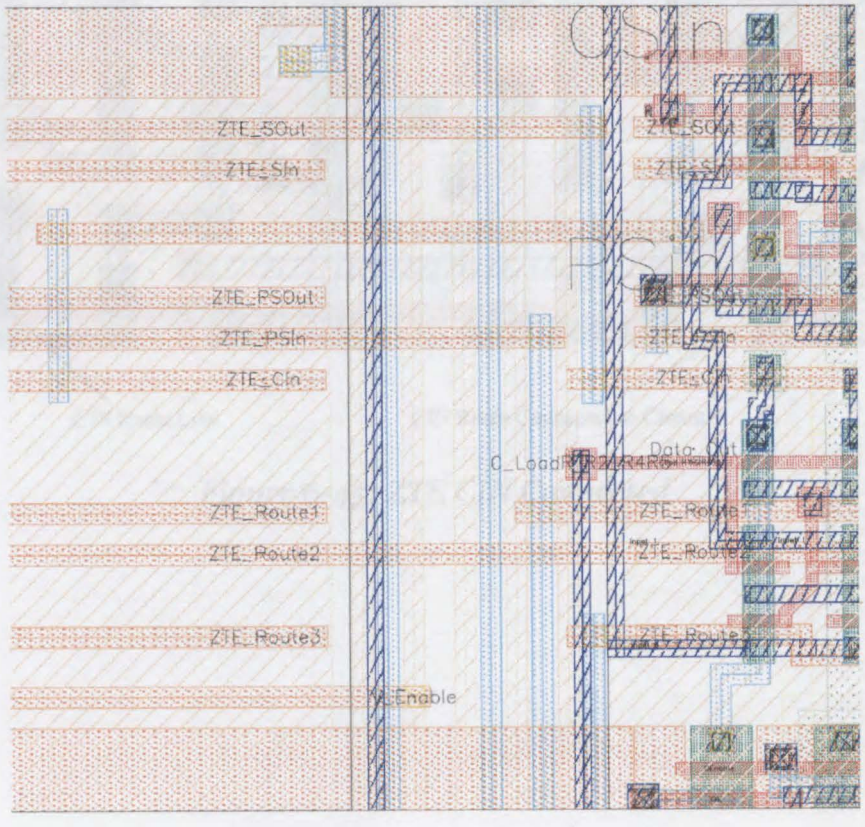


Figure 6-44: Pixel 2,2 ZTE Routes

A number of pixels, in the context of an NBLOCK, require that special connections be made to VDD or GND. For instance the lowest scale parent (pixel (1,1)) requires that it's PSIN signal be connected to VDD for the encode, and GND for decode. This is accomplished by isolating the two sections and connecting the two areas to relevant fixed statuses, for each such pixel in the NBLOCK. In addition pixels at the highest scales do not connect to the CIN status route, because they have no descendants and hence are connected to GND. This pixel-wide route line is then used for other ZTE status signals for other pixels. Figure 6-45 shows a pixel that has it CIN Route Line connected to the internal circuitry, while Figure 6-46 shows a highest scale pixel that has its internal CIN connected to ground relieving the CIN Route Line for other pixel status routes.

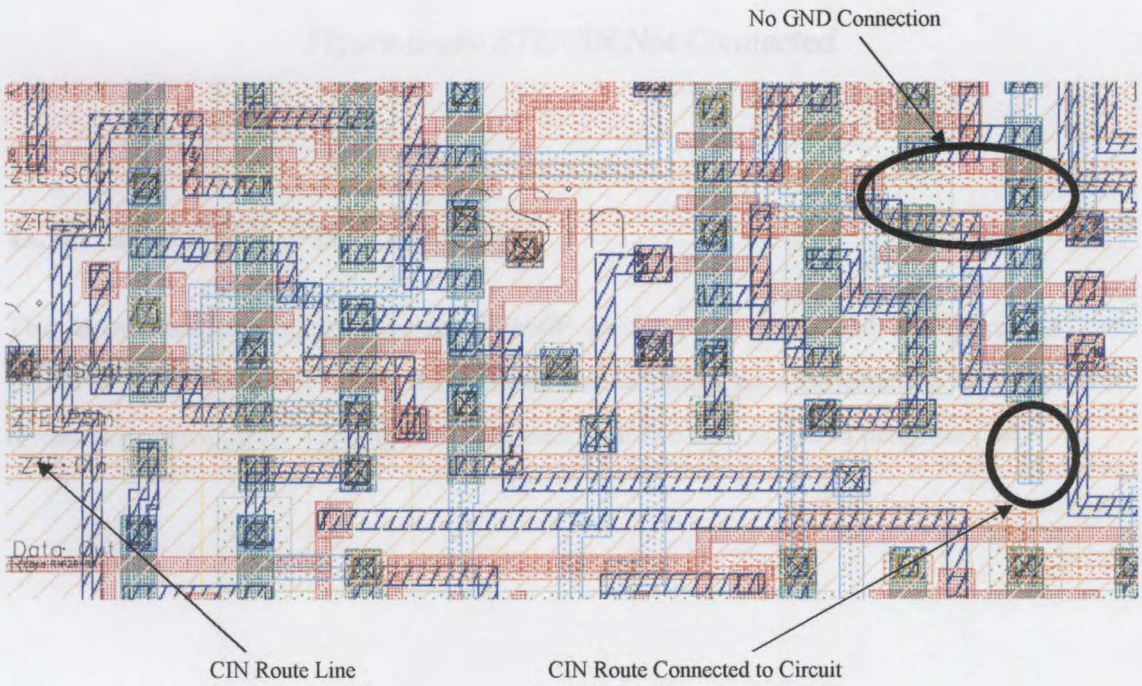


Figure 6-45: ZTE CIN Connected

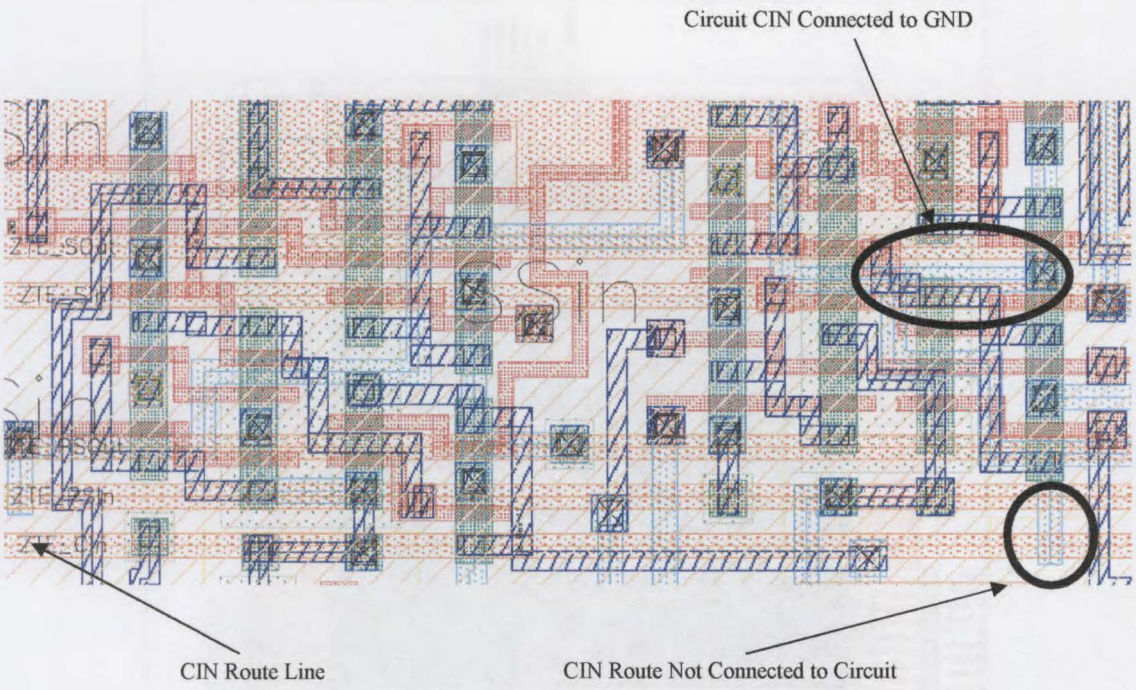


Figure 6-46: ZTE CIN Not Connected

6.9. IP100P Full Array

The layout and micrograph for the IP100P prototype is presented in Figure 6-47 and Figure 6-48 respectively. After fabrication and bonding via Europractice the chips were returned in April 2001. Operational verification of the chip is still in progress [70].

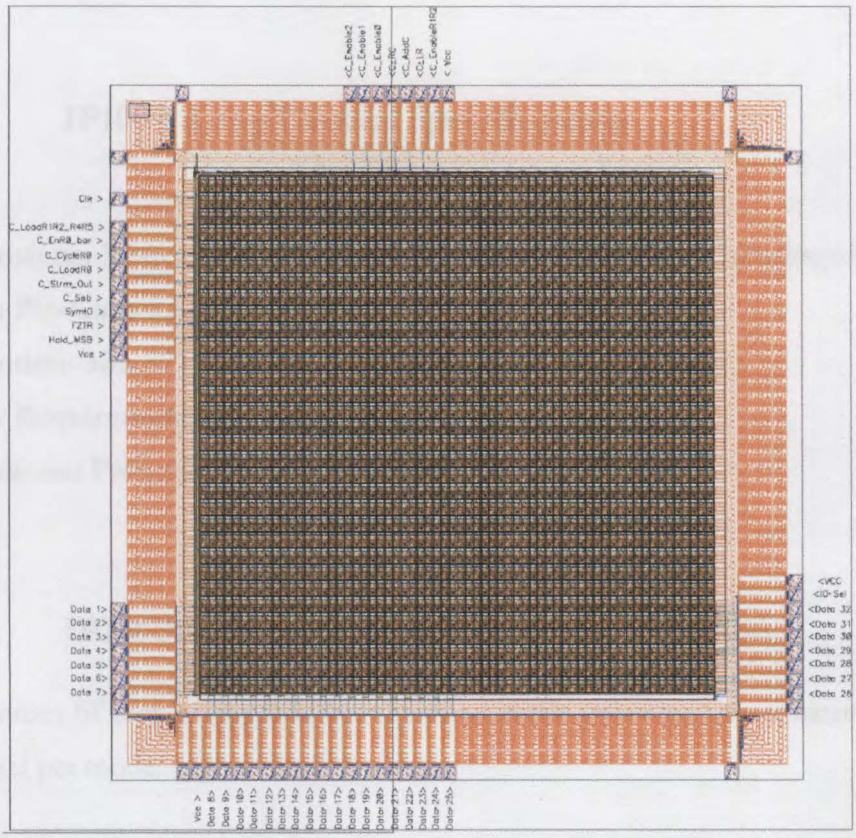


Figure 6-47: IP100P Array Layout

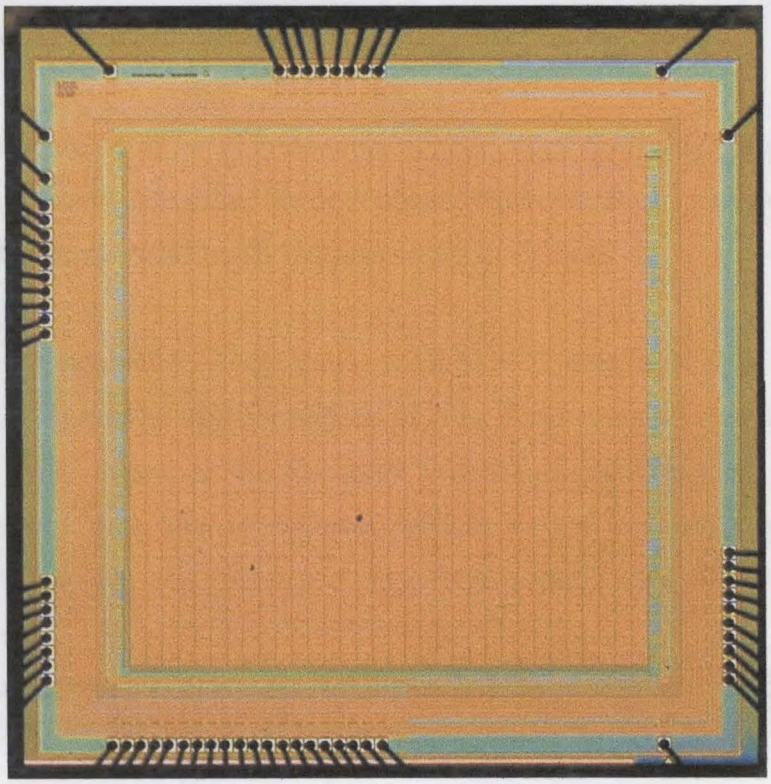


Figure 6-48: Ip100P Micrograph

6.9.1. IP100P Array General Specifications

Dimensions: 3.446 x 3.446 mm^2 (unpackaged) and 3.1 x 3.1 cm^2 (Packaged)
Active Pin-Count: 61 Pins
Resolution: 32 x 32 pixels
Power Requirements: VDD (2.5V), VCC(3.3V) & GND
Operational Frequency: 100 -- 200 KHz

6.9.2. IP100P Simulated Array Power Consumption

Simulations of each pixel mode have resulted in the following power extremes defined per pixel per mode.

Max Peak Power: 1462.29µW (ZID One Detection Phase)
Max Average Power: 1.799µW (ZTR Symbol Decode)
Min Idle Power: 1.3036 x 10^{-5}µW (DNT Detect, Latch & Shift Out)

Given these figures, a 32 by 32 array at worst case, when every pixel is searching for a non significant coefficient value, the peak power requirements can reach a maximum of 32 x 32 x 1462.29µW = **1498mW**, but for periods of (1-2)ns.

The worst case maximum average power for a 32 x 32 array is 32 x 32 x 1.799µW = **1.843mW**, which may be held for periods of 10us if the entire array is decoding ZTR symbols at once. However the maximum number of pixels that can perform this operation at one time, due to subband division, is 3 x 16 x 16 (High frequency Subbands) = 768. Therefore the maximum average power is reduced to **1.38mW**.

The minimum idle power dissipation for a 32 x 32 array is 32 x 32 x 1.3036 x 10^{-5}µW = 0.0133µW.

6.9.3. IP100P Prototype Testing

Once the IP100P prototypes had been fabricated and returned, appropriate test-beds were constructed to test the functionality of the device. The tests are currently being conducted by another group member progressing towards his research topic [70]. Initially, due to a voltage conversion issue with the test rig, the device exhibited severe data corruption when more than one pixel was accessed. After assessing the test rig further it was deemed unsuitable for testing of the IP100p chip and the testing was postponed until the installation of the new Agilant-HP tester was completed. Currently the chip's full functionality is being verified via the new tester. Preliminary results indicate the full operation of the device. This work is fully documented in [70].

6.10. Conclusion

This chapter has demonstrated the realisation feasibly of a novel parallel ZTE compression algorithm as described in Chapter 4, in 0.25 micron UMC technology. Given the current maximum die-size coupled with the size of a QCIF array, this chapter has also shown an implementation of the ZTE that is restricted to a pixel-pitch of 80 x 80μm. As a result full custom layout has been employed with all primitive components verified for functionality within the UMC technology. The operation of the ZTE component as a pixel-wise whole has been also verified with the use of Level 49 BSIM 3 HSpice Transistor model supplied by the foundry.

The significant advantage of this parallel implementation is that it performs the processing needs of a 32 x 32 x 100,000 Hz = 102.4 MHz ZTE instruction processor with a maximum average power of approximately 1.843mW. Furthermore, this implementation is easily scaled to accommodate increased resolution (i.e. QCIF, CIF, etc.), finer technology (0.13μm, 0.1μm, etc), any form of wavelet that can be implemented (i.e. Debauchies, 2,6, etc) and any number of wavelet scales (4,5 etc.).

Chapter 7

CONCLUSIONS & FURTHER RESEARCH

"I think there is a world market for maybe five computers."

Thomas Watson (1874-1956), Chairman of IBM, 1943

7.1. Contributions of this Thesis

In the interest of producing a viable product for use in mobile multimedia communications, particularly in video, this thesis has presented a novel massively parallel architecture for video compression. Initially, general aspects relating to a video compression system have been described, particularly the topics of image representation and sampling, motion compensation, transform coding, coefficient coding and entropy coding. This is followed by an in-depth analysis of the two main zerotree coding algorithms, EZW and ZTE, and associated searching algorithms. Both algorithms were shown to perform similarly and considered as ideal candidates for implementation in a video compression device. The architecture for this video compression device, the IPA, is then introduced. In order to minimise high-bandwidth component interfacing in a developed end product as well as to minimise power usage, this thesis has presented architectures for a unified image / video capture, processing and display device, the IPA. This device exploits the advantages of massively parallel processing to reduce the clock frequencies required for real-time processing. Design issues for such a 3D OPTO-VLSI device have then been discussed, particularly in reference to the parallel implementation of the wavelet and motion compensation architectures, the scale control mechanisms, high-

pass / low-pass pixel selection mechanisms and control elements. These architectures were then modified to include necessary components to support the integration of both aforementioned zerotree architectures.

Detailed descriptions of the massively parallel architectures developed for both the Embedded Zerotree Wavelet (EZW) and the ZeroTree Entropy (ZTE) coding algorithms have been provided. Solutions to a number of problems including issues relating to sequential to parallel processor conversion of the algorithms, highly parallel significance searches, per-pixel based pixel self-classification and enablement, array load / unload schemes, parallel to serial to parallel edge array conversion, and integration into the wavelet transform architecture, have been given. Furthermore, a comparison between the two zerotree architectures, in terms of coding efficiency and hardware complexity when relating to an area limited VLSI device design, has been performed in order to select the most appropriate for prototype implementation. The ZTE algorithm, due to its hardware and control simplicity was chosen and a full custom layout produced. Issues relating to power, pads and control line distribution have been discussed in addition to the full definition of the circuit in 0.25 micron UMC twin-tub technology. Simulation results for power utilisation have then been provided.

7.2. Conclusion

This thesis has presented the complete design of two novel massively parallel zerotree architectures for implementation within the Intelligent Pixel Array paradigm. In particular the ZTE has been chosen for full custom design testing in the IP100P prototype to prove the feasibility of such a product. The advantage of the proposed systems stems from the parallel nature of the design, which allows for the coding of 25 fps video sequences using a clock frequency in the region of 100 kHz, consuming less than 35mW of power when processing a QCIF image. Although the technology chosen for this implementation limits the hardware complexity of the design somewhat, the architectures and principals conveyed in this thesis represent an important stepping stone for not only future mobile multimedia communications equipment, but also an interesting area of research.

7.3. Future Research Opportunities

This area of research presents a number of additional challenges that can form the basis of future research topics. Some of these are listed below.

7.3.1. Colour Video Processing

This is a significant area of research spanning the regimes of capture, processing and display components. Colour image processing in particular introduces a number of parallel architectural challenges including, RGB to YUV conversion, sub-sampling of colour components, register reuse and device compaction. The fundamental difficulty with colour processing relates to the available space on chip and, therefore, within a pixel, as three sets of data (Y, U and V) has to be stored, processed and exchanged. New smaller feature size VLSI process technology holds the key to this area.

7.3.2. Standards Development

At present the codec employs a propriety streaming format for the exchange of video information. However, for the product to be widely applicable either a new standard has to be developed or the present stream has to be modified to suit an existing standard such as MJPEG2000, which provides for the use of a wavelet-zerotree based algorithm for video compression. This, although not a particularly exciting area of research, is considered a must for successful product deployment in multimedia communications.

7.3.3. Increased Resolution

In future versions, technology permitting, an opportunity exists to increase the pixel capture, display and processing resolution to that of CIF, DVD or even HDTV. As with the increased pixel complexity introduced with colour, this is heavily dependant on the minimum feature size of the process technology at hand.

Although the suggestions made above are directly related to image and video coding, the parallel processing array presents other novel opportunities for devices such as optical switches, multiplexers, hologram production etc. which, may be of great use in tomorrows world.

Bibliography

[1] A. F. Blackwell (1996) – "Correction: A Picture is Worth 84.1 Words", *Proc. of the First ESP Student Workshop*, pp. 15 – 22.

[2] ARIB (1999) – "Codec for Circuit Switched Multimedia Telephony Service; General Description, 3G TS 26.110 V3.o.1 Technical Specification", *3rd Generation Partnership Project.*

[3] W. Penebaker and J. Mitchell (1992) – "JPEG Still Image Compression Standard", *Van Nostrand Reinhold*, New York.

[4] G. K. Wallace (1991) – "The JPEG Still Picture Compression Standard", *Comm. ACM Vol. 34 No. 4*, pp. 31-44.

[5] K. R. Rao (1998) – "Image and Video Coding Technology and Standards", *Course Notes.* University of Western Australia.

[6] O. Edder, P. Fleury, T. Ebrahimi and M. Kunt (1999) – "High-Performance Compression of Visual Information – A Tutorial Review – Part 1: Still Pictures", *Proceedings of the IEEE Vol. 87 No. 6*, pp. 976-1008.

[7] A. Said and W. A. Pearlman (1996) – "A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees", *IEEE Trans. On Circuits and Systems for Video Technology*, Vol 6, No 3, pp. 243 – 250.

[8] A. C. Hung (1993) – "PVRG-P64 Codec 1.1", *Program Documentation*, Stanford University.

[9] D. Chen and A. C. Bovik (1990) – "Visual Pattern Image Coding", *IEEE Trans. on Communication Vol. 38 No. 12*, pp. 2137-2145.

[10] M. C. Chen and A. N. Willson (1997) – "A Spatial and Temporal Motion Vector Coding Algorithm for Low Bit-Rate Video Coding", *Proc. of ICIP'97* Vol. 2, pp. 791-794.

[11] CCITT (1990) – "Recommendation H.261: Video Codec for Audiovisual Services at P x 64 kbit/s", *Line Transmission on Non-Telephone Signals*, Geneva.

[12] MPEG-1 (1993) – "Coding of Moving Pictures and Associated Audio for Digital Storage Media Up to About 1.5 Mbit/s", *Tech. Rep., ISO/IEC IS 11172.*

[13] ITU-T Rec. H.263, (1995) "Video Coding for Low Bit-Rate Communications".

[14] MPEG-2 (1994) – "Generic Coding of Moving Pictures and Associated Audio", *Tech. Rep., ISO/IEC DIS 13818.*

[15] C. Auyeung, J. Kosmach, M. Orchard, and T. Kalafatis (1992) – "Overlapped Block Motion Estimation", *Proc. of ICVCIP'92*, pp. 561-572.

[16] M. H. Chan, Y. B. Yu, and A. G. Constantinides (1990) – "Variable Size Block Matching Motion Compensation with Applications to Video Coding", *IEE Proceedings Vol. 137 Pt. I No. 4*, pp. 205-212.

[17] M. T. Orchard and G. J. Sullivan (1994) – "Overlapped Block Motion Compensation: An Estimation-Theoretic Approach", *IEEE Trans. on Image Processing Vol. 3 No. 5*, pp. 693-699.

[18] N. Ahmed, T. Natarajan and K. R. Rao (1974) – "Discrete Cosine Transform", *IEEE Trans. On Computing*, pp. 90-93.

[19] R. J. Clarke (1985) – "Transform Coding of Images", *Academic Press*, New York.

[20] M. Costa and K. Tong (1994) – "A Simplified Integer Cosine Transform and Its Application in Image Compression", *TDA Progress Report 42-119*, pp. 129-139.

[21] Y. H. Chan and W. C. Siu (1994) – "Short Communication, An Approach to Subband DCT Image Coding", *Journal of Visual Communication and Image Representation*, pp. 95-106.

[22] J. M. Shapiro (1993) – "Embedded Image Coding Using Zerotrees of Wavelet Coefficients", *IEEE Trans. On Signal Processing Vol. 41 No. 12*, pp. 3445-3462.

[23] A. M. Rassau, G. Alagoda, D. Lucas, J. Austin-Crowe, K. Eshraghian (1999) – "Massively Parallel Intelligent-Pixel Implementation of a Zerotree Entropy Video Codec for Multimedia Communications", *VLSI: Systems On A Chip, Kluwer Academic Publishers*, Portugal, 89-100.

[24] A. M. Rassau, G. Alagoda, K. Eshraghian (1999) – "Massively Parallel Wavelet Based Video Codec For An Intelligent-Pixel Mobile Multimedia Communicator", *Fifth International Symposium on Signal Processing and its Application*, Queensland, 793-795.

[25] P. M. Bentley and J. T. E. McDonnell (1994) – "Wavelet Transforms: An Introduction", *Electronics and Communication Engineering Journal*, pp. 175-194.

[26] A. Grossman and J. Morlet (1984) – "Decomposition of Hardy Functions Into Square Integrable Wavelets of Constant Shape", *SIAM Journal of Math. Vol. 15*, pp. 723-736.

[27] Y. Meyer (1989) – "Wavelets and Operators, Analysis at Urbana vol.1 Edited by E. Berkson, N.T. Peck and J. Uhl", *Lecture Notes Series*, London Math. Society.

[28] I. Daubechies (1988) – "Orthonormal Bases of Compactly Supported Wavelets", *Commun. on Pure and Applied Mathematics Vol. XLI*, pp. 909-996.

[29] S. G. Mallat (1989) – "A Theory for Multi-resolution Signal Decomposition: The Wavelet Representation", *IEEE Trans. Pattern Analysis and Machine Intell., Vol. 11*, pp. 674-693.

[30] A. Graps, (1995) – "An Introduction to Wavelets", *IEEE Computational Science and Engineering Vol. 2 No. 2.*

[31] K. Ramchandran, K., Vetterli, M. and Herley, C., "Wavelets, Subband Coding and Best Bases", *Proc. of IEEE Vol. 84 No. 4.*

[32] R. J. Clarke (1996) – "Digital Compression of Still Images and Video", *Academic Press Ltd.*, London.

[33] E. P. Simoncelli and E. H. Adelson (1990) – "Subband Coding: Chapter 4 Subband Transforms edited by J. Woods", *Kluwer Academic Press*, pp. 143-192.

[34] Z. Xiong, O. Guleryuz and M. T. Orchard (1996) – "A DCT-based Embedded Image Coder", *IEEE Signal Processing Letters Vol. 3 No. 11*, pp. 289-290.

[35] O. Rioul and M. Vetterli (1991) – "Wavelets and Signal Processing", *IEEE SP Magazine*, pp. 14-38.

[36] S. G. Mallat (1989) – "Multiresolution Approximations and Wavelet Orthonormal Bases in L^2R", *Trans. of the American Math. Society Vol. 315 No. 1*, pp. 69-87.

[37] R. A. Gopinath and C. S. Burrus (1992) – "Wavelet Transforms and Filter Banks", *Wavelets: A Tutorial in Theory and Applications, C. K. Chui, ed*, Academic Press, San Diego.

[38] H. Caglar, Y. Liu and A. N. Akansu (1993) – "Optimal PR-QMF Design for Subband Image Coding", Proc. of Journal of Visual Communication and Image Representation Vol. 4 No.3, Academic Press Inc., pp. 242-253.

[39] G. Strang and T. Nguyen (1996) – "Wavelets and Filter Banks", *Wellesley Cambridge Press.*

[40] J. D. Villasenor, B. Belzer, and J. Liao (1995) – "Wavelet Filter Evaluation for Image Compression", *IEEE Trans. On Image Processing Vol. 4, No. 8*, pp. 1053-1060.

[41] J. Y. Tham, S. Ranganath and A. A. Kassim (1998) – "Highly Scalable Wavelet-Based Video Codec for Very Low Bit-Rate Environment", *IEEE Journal on Selected Areas in Communication Vol. 16 No. 1*, pp. 12-27.

[42] D. Marpe and H. L. Cycon (1999) – "Very Low Bit-Rate Video Coding Using Wavelet-Based Techniques", IEEE Trans. on Circuits and Systems for Video Technology Vol. 9 No. 1, pp. 85-94.

[43] Y. H. Kim and J. Modestino (1992) – "Adaptive Entropy Coded Subband Coding of Images", *IEEE Trans. Image Processing Vol. 1*, pp. 31-48.

[44] A. M. Rassau (1999) – "Massively Parallel Wavelet Based Video Codec for an Intelligent-Pixel Mobile Multimedia Communicator", *Ph.D. Thesis Department of Cybernetics*, University of Reading, England.

[45] G. Knowles (1990) – "VLSI Architecture for the Discrete Wavelet Transform", *Electronics Letters Vol. 26*, pp. 1184-1185.

[46] D. A. Huffman (1952) – "A Method for the Construction of Minimum Redundancy Codes", *Proc. of the Institute of Radio Engineers Vol. 40*, pp. 1098-1101.

[47] F. Halsall (1995) – "Data Communications, Computer Networks and Open Systems (Fourth Edition)", Addison-Wesley Publishing Company Inc., USA, pp.139-145.

[48] D. S. Hirshberg and D. A. Lelewer (1990) – "Efficient Decoding of Prefix Codes", *Comm. ACM Vol. 33 No. 4*, pp. 449-459.

[49] I. H. Witten, R. M. Neal and J. G. Cleary (1987) – "Arithmetic Coding for Data Compression", *Computing Practices Communications of the ACM Vol. 30 No. 6*, pp. 520-540.

[50] C. E. Shannon (1948) – "A Mathematical Theory of Communication", *Bell Systems Tech. Journal Vol. 27*, pp 379-423, 623-656.

[51] H. Man, F. Kossentini and M. T. J. Smith (1997) – "Robust EZW Image Coding for Noisy Channels", *Proc. of IEEE Signal Processing Letters Vol. 4 No. 8*, pp. 227-229.

[52] J. M. Zhong, C. H. Leung and Y. Y. Tang (2000) – "An Improved Embedded Zerotree Wavelet Image Coding Method Based on Coefficient Partitioning Using Morphological Operation", *Journal of Pattern Recognition and Artificial Intelligence Vol. 14 No. 6*, pp. 795-807.

[53] E. Kang, T Tanaka and S. Ko (1999) – "Improved Embedded Zerotree Wavelet Coder", *Electronics Letters Vol. 35 No. 9*, pp. 705-706.

[54] S. A. Martucci, I. Sodagar (1996) – "Zerotree Entropy Encoding of Wavelet Coefficients for Very Low Bit-Rate Video", *Proc. of the IEEE Int. Conf. Image Processing*, Switzerland

[55] S. A. Martucci, I. Sodagar, T. Chiang and Y. Zhang (1997) – "A Zerotree Wavelet Video Coder", *IEEE Trans. on Circuits and Systems for Video Technology Vol. 7 No. 1*, pp. 109-118.

[56] A. Kaup (1999) – "Object-based Texture Coding of Moving Video in MPEG4", Proc. IEEE Trans. on Circuits and System for Video Technology Vol 9, pp. 5-15.

[57] T. Cormen, C. Leiserson, and R. Rivest (1994) – "Introduction to Algorithms" (14th printing). *The MIT Press.*

[58] A. M. Rassau, K. Eshraghian, H. Cheung, S. W. Lachowicz, T. C. B. Yu, W. A. Crossland and T. D. Wilkinson (1998) – "Smart Pixel Implementation of a 2-D Parallel Nucleic Wavelet Transform for Mobile Multimedia Communications", *Proc. of the Design, Automation and Test in Europe Conference.*

[59] C. Spizig (2000) – "Analog-to-Digital Converter Array Implementation for the Camera-On-A-CMOS Chip", *Masters thesis: University of Ulm and Edith Cowan University.*

[60] W. A. Crossland, T. D. Wilkinson, T. M. Coker, T. C. B. Yu and M. Stanley (1997) – "The Fast Bitplane SLM: A New Ferroelectric Liquid Crystal on Silicon Spatial Light Modulator Designed for High Yield and Low Cost Manufacturability", *OSA TOPS Vol. 14 Spatial Light Modulators*, pp. 102-106.

[61] N. Collings, W. A. Crossland, P. J. Ayliffe, D. G. Vass, and I. Underwood (1989) – "Evolutionary Development of Advanced Liquid Crystal Spatial Light Modulators", *Applied Optics Vol. 28 No. 22.*

[62] L. Díaz de Cerio, A. González and M. Valero-García (1996) – "Communication Pipelining in Hypercubes", *Parallel Processing Letters Vol. 6 No. 4*, pp. 507-523.

[63] S. Y. Lee and J. K. Aggarwal (1986) – "Parallel 2-D Convolution on a Mesh Connected Array Processor" *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 299-304.

[64] L. Wall, K. Ferens and W.Kinser (1993) – "Real-Time Dynamic Arithmetic Coding for Low Bit-Rate Channels", *Proc. IEEE Communications, Computers & Power Conference*, pp. 381-391.

[65] H. N. Cheung, G. Alagoda, K. Eshraghian and L. Ang (1999) – "Smart Pixel VLSI Architecture For Embedded Zerotree Wavelet Coding", *Fifth International Symposium on Signal Processing and its Application*, Queensland, 693-695.

[66] J. Bae and V. K. Prasanna (1995) – "A Fast and Area-efficient VLSI Architecture for Embedded Image Coding", *Proc. of International Conference on Image Processing, Vol. 3*, pp. 452-455.

[67] L. Ang, H. N. Cheung, and K. Eshraghian (1998) "VLSI Architecture for Significance Map Coding of Embedded Zerotree Wavelet Coefficients", *Proc of APCCAS'98*, pp. 627-630.

[68] K. Eshraghian (1991) – "CMOS & BiCMOS VLS Design Analog & Digital", *Intensive Summer Course at the Swiss Federal Institute of Technology*, Lausanne Electronics Laboratories.

[69] D. P. Pucknell and K. Eshraghian (1994) – "Basic VLSI Design Third Edition", *Silicon Systems Engineering Series*, Prentice Hall, Australia.

[70] D. Lucas (2001-2002) – "Primitives and Design of the Intelligent Pixel Multimedia Communicator", *Ph. D. Thesis, Edith Cowan University*, Still in progress.

[71] N. H. E. Weste and K. Eshraghian (1993) – "Principals of CMOS VLSI Design, A Systems Perspective, Second Edition", *Addison-Wesley Publishing Company*.

[72] P. Pirsch, N. Demassieux and W. Gehrke (1995) – "VLSI Architectures for Video Compression – A Survey", *Proc. of the IEEE Vol. 83 No. 2*, pp. 220-246.

Appendix A: EZW VHDL Listings

```
------------------------------
-- IP 1 Bit Register
-- G.Alagoda 2001
-- Rev. 001
------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY REG1 IS
    PORT(
        DI              : in std_logic;
        CLR, CLK        : in std_logic;
        DO, DOB         : out std_logic
        );
END REG1;

ARCHITECTURE STRUCT OF REG1 IS

signal A, B, C, D, E, F       : std_logic;

BEGIN
  A <= DI when (CLK = '0') else B;
  B <= (not C);
  C <= (A nand CLR);
  D <= C when (CLK = '1') else F;
  E <= (not D);
  F <= (E nand CLR);
  DO <= E;
  DOB <= F;
END STRUCT;

------------------------------
-- IP R0 Register Component
-- G.Alagoda 2001
-- Rev. 001
------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY MAINREG IS
    PORT(
        CLK, EnR0, ClkMSB, R0Ctl, R0Ct2             : in std_logic;              --
Global Control In
        HPRin, Above, RC                    : in std_logic;        --    Other    Mixed
Signals
        GEnable, WT, Rev, SigE, ZR01        : in std_logic;             --    Internal
Control In
        Adder, EXT, R4out                   : in std_logic;             -- Data Paths
In
        ConVa, EnR0a                        : out std_logic;       --  Internal  Contol
Out
        R0out, Sign                         : out std_logic             -- Data Paths
Out
        );
END MAINREG;

ARCHITECTURE STRUCT OF MAINREG IS

COMPONENT REG1 IS
    PORT(
        DI              : in std_logic;
        CLR, CLK        : in std_logic;
        DO, DOB         : out std_logic
        );
END COMPONENT;

signal MXOV             : std_logic_vector (12 downto 1);
signal ROUTV            : std_logic_vector (8 downto 0);
signal ROUTBV           : std_logic_vector (8 downto 0);
signal T1, T2, T3, T4   : std_logic;                                        --       Temp
signals
```

```
signal CLR              : std_logic := '1';
signal LP               : std_logic;

BEGIN
-- Add reg elements
  L0 : REG1 port map (DI => MXOV(10),CLR => CLR,CLK => T1,DO => ROUTV(0),DOB =>
ROUTBV(0));
  L1 : REG1 port map (DI => ROUTV(2),CLR => CLR,CLK => T1,DO => ROUTV(1),DOB =>
ROUTBV(1));
  L2 : REG1 port map (DI => ROUTV(3),CLR => CLR,CLK => T1,DO => ROUTV(2),DOB =>
ROUTBV(2));
  L3 : REG1 port map (DI => ROUTV(4),CLR => CLR,CLK => T1,DO => ROUTV(3),DOB =>
ROUTBV(3));
  L4 : REG1 port map (DI => ROUTV(5),CLR => CLR,CLK => T1,DO => ROUTV(4),DOB =>
ROUTBV(4));
  L5 : REG1 port map (DI => MXOV(9),CLR => CLR,CLK => T1,DO => ROUTV(5),DOB =>
ROUTBV(5));
  L6 : REG1 port map (DI => MXOV(8),CLR => CLR,CLK => T1,DO => ROUTV(6),DOB =>
ROUTBV(6));
  L7 : REG1 port map (DI => MXOV(7),CLR => CLR,CLK => T1,DO => ROUTV(7),DOB =>
ROUTBV(7));
  L8 : REG1 port map (DI => MXOV(5),CLR => CLR,CLK => T2,DO => ROUTV(8),DOB =>
ROUTBV(8));

-- Clocks
  T1 <= (CLK and GEnable and T4);
  T2 <= (CLK and ClkMSB and GEnable);

-- Muxes
  MXOV(1) <= MXOV(2) when (LP = '1') else Adder;
  MXOV(2) <= R4out when (ZRO1 = '1') else ROUTV(0);
  MXOV(3) <= EXT when (R0Ct1 = '0') else MXOV(1);
  MXOV(4) <= MXOV(2) when ((R0Ct1 nor R0Ct2) = '1') else MXOV(3);
  MXOV(5) <= R4out when (SigE = '1') else MXOV(4);
  MXOV(6) <= ROUTV(8) when (MT = '1') else MXOV(5);
  MXOV(7) <= ROUTV(7) when (Rev = '1') else MXOV(11);
  MXOV(8) <= MXOV(2) when (Rev = '1') else ROUTV(7);
  MXOV(9) <= ROUTV(1) when (Rev = '1') else ROUTV(6);
  MXOV(10) <= ROUTV(6) when (Rev = '1') else ROUTV(1);
  MXOV(11) <= SigE when (SigE = '1') else MXOV(6);
  MXOV(12) <= Above when (RC = '1') else HPRin;

-- General Crap
  T3 <= (R0Ct2 nor (not R0Ct1));
  ConVa <= T3;
  R0out <= ROUTV(0);
  Sign <= ROUTV(8);
  T4 <= EnR0 or (MT and MXOV(12));
  EnR0a <= T4;
  LP <= (MT and (not MXOV(12)) and (not T3)) or ((not T3) nor (ROUTV(8)));


END STRUCT;

-------------------------------
-- IP Adder Component
-- G.Alagoda 2001
-- Rev. 001
-------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY ADDER IS
    PORT(
        CLK             : in std_logic;
        R0, MCEXT       : in std_logic;
        Sout            : OUT std_logic;
        S, EnR0a, ConVa         : IN std_logic
        );
END ADDER;

ARCHITECTURE STRUCT OF ADDER IS

COMPONENT REG1 IS
    PORT(
        DI              : in std_logic;
        CLR, CLK        : in std_logic;
```

```
        DO, DOB         : out std_logic
        );
END COMPONENT;

signal M1       : std_logic;
signal Cout     : std_logic;
signal Cin      : std_logic := '0';
signal A1, A2   : std_logic;
signal CLR      : std_logic := '1';
signal DOB      : std_logic;

BEGIN
-- Add A single reg element
  L1 : REG1 port map (DI => M1,CLR => CLR,CLK => CLK,DO => Cin,DOB => DOB);
-- Add
  A1 <= (R0 xor ConVa);
  A2 <= ((MCEXT xor S) nor ConVa);
  Sout <= (A1 xor A2) when Cin = '0' else (A1 xnor A2);
  Cout <= (A1 and A2) or (Cin and A1) or (Cin and A2);
  M1 <= Cout when EnR0a = '1' else ((not S) or ConVa);
END STRUCT;

-------------------------------
-- IP Motion Compensation Component
-- G.Alagoda 2001
-- Rev. 001
-------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY MCM IS
    PORT(
        CLK, LR, LR1245, Strmout            : in std_logic;             --      Global
Control In
        MC                                  : in std_logic;             --    Internal
Control In
        Adder, R0out                        : in std_logic;             -- Data Paths
In
        MCout                               : out std_logic             -- Data Paths
Out
        );
END MCM;

ARCHITECTURE STRUCT OF MCM IS

COMPONENT REG1 IS
    PORT(
        DI          : in std_logic;
        CLR, CLK    : in std_logic;
        DO, DOB     : out std_logic
        );
END COMPONENT;
signal CLR      : std_logic := '1';

signal MX           : std_logic_vector (3 downto 0);
signal R1O          : std_logic_vector (5 downto 0);
signal R2O          : std_logic_vector (5 downto 0);
signal R1OB         : std_logic_vector (5 downto 0);
signal R2OB         : std_logic_vector (5 downto 0);
signal T1, T2, T3   : std_logic;                                        --      Temp
signals


BEGIN
-- Add reg1 elements
  L10 : REG1 port map (DI => R1O(1),CLR => CLR,CLK => T1,DO => R1O(0),DOB => R1OB(0));
  L11 : REG1 port map (DI => R1O(2),CLR => CLR,CLK => T1,DO => R1O(1),DOB => R1OB(1));
  L12 : REG1 port map (DI => R1O(3),CLR => CLR,CLK => T1,DO => R1O(2),DOB => R1OB(2));
  L13 : REG1 port map (DI => R1O(4),CLR => CLR,CLK => T1,DO => R1O(3),DOB => R1OB(3));
  L14 : REG1 port map (DI => R1O(5),CLR => CLR,CLK => T1,DO => R1O(4),DOB => R1OB(4));
  L15 : REG1 port map (DI => MX(2),CLR => CLR,CLK => T1,DO => R1O(5),DOB => R1OB(5));

-- Add reg2 elements
  L20 : REG1 port map (DI => R2O(1),CLR => CLR,CLK => T2,DO => R2O(0),DOB => R2OB(0));
  L21 : REG1 port map (DI => R2O(2),CLR => CLR,CLK => T2,DO => R2O(1),DOB => R2OB(1));
  L22 : REG1 port map (DI => R2O(3),CLR => CLR,CLK => T2,DO => R2O(2),DOB => R2OB(2));
  L23 : REG1 port map (DI => R2O(4),CLR => CLR,CLK => T2,DO => R2O(3),DOB => R2OB(3));
```

```
  L24 : REG1 port map (DI => R2O(5),CLR => CLR,CLK => T2,DO => R2O(4),DOB => R2OB(4));
  L25 : REG1 port map (DI => MX(1),CLR => CLR,CLK => T2,DO => R2O(5),DOB => R2OB(5));

-- Clocks
  T3 <= (CLK and MC);
  T1 <= (T3 and (not Strmout));
  T2 <= (T3 and Strmout);

-- Muxes
  MX(3) <= Adder when (LR = '1') else R0out;
  MX(2) <= MX(3) when (LR1245 = '1') else R1O(0);
  MX(1) <= MX(3) when (LR1245 = '1') else R2O(0);
  MX(0) <= R2O(0) when (Strmout = '1') else R1O(0);

-- General Crap
  MCout <= MX(0);
END STRUCT;

------------------------------
-- IP Significance identification Component
-- For the EZW
-- G.Alagoda 2001
-- Rev. 001
------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY SIDM IS
    PORT(
        CLK, Strmout, FZT                    : in std_logic;              --      Global
Control In
        ZT, EnR0a                            : in std_logic;              --    Internal
Control In
        Cin, Sbin, R0out, R4out, ZS3, ZRO3   : in std_logic;              -- Data Paths
In
        SigE, Sig, ZS1, ZS2, Sbout, Pout     : out std_logic              -- Data Paths
Out
        );
END SIDM;

ARCHITECTURE STRUCT OF SIDM IS

COMPONENT REG1 IS
    PORT(
        DI              : in std_logic;
        CLR, CLK        : in std_logic;
        DO, DOB         : out std_logic
        );
END COMPONENT;
signal CLR      : std_logic := '1';

signal R3O, R3OB              : std_logic;
signal T1, T2, T3, T4, T5, T6 : std_logic;                                    --
Temp signals


BEGIN
-- Add reg element
  L1 : REG1 port map (DI => T1,CLR => CLR,CLK => T2,DO => R3O,DOB => R3OB);

-- Clocks
  T2 <= (CLK and (ZT nand ((EnR0a or ((not Strmout) and FZT)) nand R3OB)));  --

-- General Crap
  SigE <= R3OB and T1;
  Sig <= R3O;
  T1 <= (ZT and R0out and Strmout) or (ZT and R4out and (not Strmout));
  T3 <= R3OB and T1;
  ZS1 <= T3 and Strmout;
  ZS2 <= (Strmout and ((Cin nand (not T3)) nand (ZRO3 nand T3)));
  T4 <= Cin or Sbin;
  Sbout <= T4 or T3;
  Pout <= T4 when (Strmout = '1') else ZS3;
END STRUCT;

------------------------------
```

```
-- IP EZWM Component
-- G.Alagoda 2001
-- Rev. 001
-------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY EZWM IS
    PORT(
        CLK, LR1245, Strmout, FZT, HPRIn, VEnable , HEnable, S      :    in    std_logic;
             -- Global Control In
        EnR0a, ZT                                                   : in std_logic;
        -- Internal Control In
        Pin, Sbin, Cin, ZRO3, R0out, EXT                            : in std_logic;
        -- Data Paths In
        GEnable, HPROut                                                  :    out   std_logic;
        -- Internal Control Out
        Sbout, Pout, Pixout, SigE, ZRO1                                  :    out   std_logic;
        -- Data Paths Out
        Abovei, Belowi                                              : in std_logic;
        -- Unidirectional ??
        Aboveo, Belowo                                              : out std_logic
        -- Unidirectional ?
        );
END EZWM;

ARCHITECTURE STRUCT OF EZWM IS

COMPONENT REG1 IS
    PORT(
        DI              : in std_logic;
        CLR, CLK        : in std_logic;
        DO, DOB         : out std_logic
        );
END COMPONENT;

COMPONENT SIDM IS
    PORT(
        CLK, Strmout, FZT                   : in std_logic;             --      Global
Control In
        ZT, EnR0a                           : in std_logic;             --    Internal
Control In
        Cin, Sbin, R0out, R4out, ZS3, ZRO3  : in std_logic;             -- Data Paths
In
        SigE, Sig, ZS1, ZS2, Sbout, Pout    : out std_logic             -- Data Paths
Out
        );
END COMPONENT;

signal MXOV                                             : std_logic_vector (1 to 10);
signal R4out, R5out, R6out, R4outb, R5outb, R6outb      : std_logic;
signal T1, T2, T3, T4                                   : std_logic;            --        Temp
signals
signal CLR                                              : std_logic := '1';
signal Sig                                              : std_logic;
signal ZS1, ZS2, ZS3                                    : std_logic;

BEGIN
-- Add Sig Component
  SIDM1 : SIDM port map (CLK => CLK, Strmout => Strmout, FZT => FZT, ZT => ZT, EnR0a =>
EnR0a,
        Cin => Cin, Sbin => Sbin, R0out => R0out, R4out => R4out, ZS3 => ZS3, ZRO3 =>
ZRO3,
        SigE => SigE, Sig => Sig, ZS1 => ZS1, ZS2 => ZS2, Sbout => Sbout, Pout => Pout);

-- Add reg elements
  R4 : REG1 port map (DI => MXOV(2),CLR => CLR,CLK => T1,DO => R4out,DOB => R4outb);
  R5 : REG1 port map (DI => MXOV(3),CLR => CLR,CLK => T1,DO => R5out,DOB => R5outb);
  R6 : REG1 port map (DI => MXOV(9),CLR => CLR,CLK => T2,DO => R6out,DOB => R6outb);

-- Clocks
  T1 <= T3 and CLK and (EnR0a nor (not ZT));
  T2 <= (FZT and CLK);

-- Muxes
  MXOV(1) <= R0out when ((T4 and (not FZT) and Strmout) = '1') else (EXT);
  MXOV(2) <= MXOV(1) when (LR1245 = '1') else (ZS1);
```

```
  MXOV(3) <= R4out when (LR1245 = '1') else (ZS2);
  MXOV(4) <= MXOV(3) when (T4 = '1') else (R5out);
  MXOV(5) <= R0out when (EnR0a = '1') else MXOV(4);
  MXOV(6) <= (MXOV(5)) when (T3 = '1') else (EXT);
  MXOV(7) <= (Sig) when (S = '1') else (Pin);
  MXOV(8) <= (Abovei) when ((Strmout or (not ZT)) = '1') else (Belowi);
  MXOV(9) <= (MXOV(8)) when (LR1245 = '1') else (T3);
  MXOV(10) <= (R6out) when ((T3 and ZT) = '1') else (T3 xor MXOV(8));

  T4 <= S and ZT;
  HPROut <= (HPRIn xor T3);
  T3 <= VEnable and HEnable and ((not ZT) or MXOV(7) or (ZT and EnR0a));
  GEnable <= T3;
  Pixout <= not(not MXOV(6));
  Aboveo <= (not (not MXOV(10))) when ((Strmout or (not ZT)) = '0') else Abovei;
  Belowo <= (not (not MXOV(10))) when ((Strmout or (not ZT)) = '1') else Belowi;
  ZS3 <= (R4outb nand R5outb);
  ZR01 <= (T4 and (not Strmout));
END STRUCT;

------------------------------
-- IP EZW Full Pixel
-- G.Alagoda 2001
-- Rev. 001
------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY IP IS
    PORT(
        CLK, LR1245, Strmout, FZT, S                    : in std_logic;
        -- Global Control In
        EnR0, LR, RC, M1, M2, ClkMSB, R0Ct1, R0Ct2, Rev          :    in    std_logic;
            -- Global Control In
        Pin, Sbin, Cin                                  : in std_logic;
        -- Data Paths In
        Lin, Rin, Uin, Bin                              : in std_logic;
        -- Data Paths In
        Abovei, Belowi, HPRIn, VEnable , HEnable        : in std_logic;
        -- Unidirectional ??
        HPROut                                          : out std_logic;      --
External Control Out
        Sbout, Pout, Pixout                             : out std_logic;      --
Data Paths Out
        Aboveo, Belowo                                  : out std_logic
        -- Unidirectional ??
        );
END IP;

ARCHITECTURE STRUCT OF IP IS
COMPONENT MCM IS
    PORT(
        CLK, LR, LR1245, Strmout                        : in std_logic;
        -- Global Control In
        MC                                              : in std_logic;
        -- Internal Control In
        Adder, R0out                                    : in std_logic;
        -- Data Paths In
        MCout                                           : out std_logic
        -- Data Paths Out
        );
END COMPONENT;
COMPONENT ADDER IS
    PORT(
        CLK                                             : in std_logic;
        R0, MCEXT                                       : in std_logic;

        Sout                                            : OUT std_logic;
        S, EnR0a, ConVa                                       : IN std_logic

        );
END COMPONENT;
COMPONENT MAINREG IS
    PORT(
        CLK, EnR0, ClkMSB, R0Ct1, R0Ct2                          :    in    std_logic;
            -- Global Control In
```

```
        HPRIn, Above, RC                                    : in std_logic;      --
Other Mixed Signals
        GEnable, WT, Rev, SigE, ZRO1                        : in std_logic;
        -- Internal Control In
        Adder, EXT, R4out                                   : in std_logic;
        -- Data Paths In
        ConVa, EnR0a                                        : out std_logic;     --
Internal Contal Out
        R0out, Sign                                         : out std_logic
        -- Data Paths Out
        );
END COMPONENT;
COMPONENT EZWM IS
    PORT(
        CLK, LR1245, Strmout, FZT, HPRIn, VEnable , HEnable, S     :    in    std_logic;
            -- Global Control In
        EnR0a, ZT                                           : in std_logic;
        -- Internal Control In
        Pin, Sbin, Cin, ZRO3, R0out, EXT                    : in std_logic;
        -- Data Paths In
        GEnable, HPROut                                            :   out   std_logic;
        -- Internal Contal Out
        Sbout, Pout, Pixout, SigE, ZRO1                            :   out   std_logic;
        -- Data Paths Out
        Abovei, Belowi                                      : in std_logic;
        -- Unidirectional ??
        Aboveo, Belowo                                      : out std_logic
        -- Unidirectional ??
        );
END COMPONENT;


signal EnR0a, SigE, ZRO1, ZRO2, Sign                    :std_logic;
signal GEnable, WT, MC, ZT, SH, ConVa                          :std_logic;
signal R0out, ADDER1, EXT, MCout, R4out, R5out          :std_logic;
signal M4                                               :std_logic_vector (1 to 3);          -- Temp
signals
signal Sig, HPMUX, MCEXT                                :std_logic;

BEGIN
-- Component Port Maps (Sheesh)
ADD1: ADDER port map (CLK => CLK, R0 => R0out, MCEXT => MCEXT, Sout => ADDER1, S => S, EnR0a => EnR0a,
ConVa => ConVa);
MCM1: MCM port map      (CLK => CLK, LR => LR, LR1245 => LR1245, Strmout => Strmout,
                        MC => MC, Adder => ADDER1, R0out => R0out, MCout => MCout);
MRM1: MAINREG port map  (CLK, EnR0, ClkMSB, R0Ct1, R0Ct2, HPRIn, Abovei, RC, GEnable, WT, Rev, SigE,
                        ZRO1, ADDER1, EXT, R4out, ConVa, EnR0a, R0out, Sign);
EZW1: EZWM port map (CLK, LR1245, Strmout, FZT, HPRIn, VEnable , HEnable, S,
                        EnR0a, ZT, Pin, Sbin, Cin, Sign, R0out, EXT, GEnable, HPROut,
                        Sbout, Pout, Pixout, SigE, ZRO1, Abovei, Belowi,
                        Aboveo, Belowo);

-- Muxes
 M4(1) <= Rin when (LR = '1') else Lin;
 M4(2) <= Bin when (LR = '1') else Uin;
 M4(3) <= M4(2) when (RC = '1') else M4(1);
 EXT <= '0' when ((SH = '1') and (ClkMSB = '1')) else M4(3);
 HPMUX <= Abovei when (RC = '1') else HPRIn;
 MCEXT <= MCout when (MC = '1') else EXT;

-- Logic Me Baby
 SH <= '1' when ((M1 = '0') and (M2 = '0')) else '0';
 MC <= '1' when ((M1 = '0') and (M2 = '1')) else '0';
 WT <= '1' when ((M1 = '1') and (M2 = '0')) else '0';
 ZT <= '1' when ((M1 = '1') and (M2 = '1')) else '0';

END STRUCT;
```

```
-----------------------------
-- Ip Test bad
-- Simple Decode
-- G.Alagoda
-----------------------------
library ieee ;
use ieee.std_logic_1164.all;

use std.textio.all;
use ieee.std_logic_textio.all;

ENTITY IP_TB_EZWDEC is

End IP_TB_EZWDEC ;

ARCHITECTURE STIM of IP_TB_EZWDEC IS

-- Time Periods
   constant PERIOD : time := 1000 ns;

-- Add Some Components
COMPONENT IP IS
    PORT(
        CLK, LR1245, Strmout, PZT, S                          : in std_logic;
        -- Global Control In
        EnRO, LR, RC, M1, M2, ClkMSB, ROCt1, ROCt2, Rev              :    in     std_logic;
              -- Global Control In
        Pin, Sbin, Cin                                        : in std_logic;
        -- Data Paths In
        Lin, Rin, Uin, Bin                                    : in std_logic;
        -- Data Paths In
        Abovei, Belowi, HPRIn, VEnable , HEnable              : in std_logic;
        -- Unidirectional ??
        HPROut                                                : out std_logic;       --
External Control Out
        Sbout, Pout, Pixout                                   : out std_logic;       --
Data Paths Out
        Aboveo, Belowo                                        : out std_logic
        -- Unidirectional ??
        );
END COMPONENT;

-- Signal Definitions
signal CLK              : std_logic := '0';
signal Cnt              : integer := 0;
signal Fin              : std_logic := '0';
signal Tdata            : std_logic_vector (8 downto 1);


signal  LR1245, Strmout, PZT, HPRIn, VEnable , HEnable, S   : std_logic := '0';
        -- Global Control In
signal  EnRO, LR, RC, M1, M2, ClkMSB, ROCt1, ROCt2, Rev            : std_logic  := '0';
              -- Internal Control In
signal  Pin, Sbin, Cin                                      : std_logic := '0';
        -- Data Paths In
signal  Lin, Rin, Uin, Bin                                  : std_logic := '0';
        -- Internal Contel Out
signal  HPROut, Sbout, Pout, Pixout                         : std_logic;
        -- Data Paths Out
signal  Abovei, Belowi                                      : std_logic := '0';
        -- Unidirectional ??
signal  Aboveo, Belowo                                      : std_logic;
        -- Unidirectional ??

--signal Idx1                                               : integer := 8;

signal OB                                                   :    std_logic_vector    (16
downto 1) := "0000000000000000";
shared variable Idx1, Idx2                                  : integer := 0;


procedure WP (variable Mult : integer) is
BEGIN
   wait for PERIOD*Mult;
END WP;
```

```
BEGIN
-- Do Port Map (Sheesh)
IP01 : IP port map (CLK, LR1245, Strmout, FZT, S, EnR0, LR, RC, M1, M2, ClkMSB, R0Ct1,
R0Ct2, Rev,
                    Pin, Sbin, Cin, Lin, Rin, Uin, Bin, Abovei, Belowi, HPRIn, VEnable
, HEnable,
                    HPROut, Sbout, Pout, Pixout,
                    Aboveo, Belowo);

-- Set Clock and other concurrent crap
   CLK <= ((not CLK) and (not Fin)) after (PERIOD/2);
   Cnt <= Cnt + 1 after PERIOD when (Fin = '0');


T1 : process begin

-- Do Test process Here

-- Clear significance info
-- Test Load and Subtract
-- Clear all three regs with zeros

VEnable <= '1'; HEnable <= '1';
LR1245 <= '0';
Strmout <= '0';
FZT <= '0';
S <= '0';
EnR0 <= '1';
ClkMSB <= '1';
Rev <= '0';
LR <= '0'; RC <= '0';
M1 <= '0'; M2 <= '0';
R0Ct1 <= '0'; R0Ct2 <= '1';
Pin <= '0'; Cin <= '0'; Sbin <= '0';
Lin <= '0'; Rin <= '0'; Uin <= '0'; Bin <= '0';
Abovei <= '0'; Belowi <= '0';
Wait for PERIOD;

M1 <= '1';
M2 <= '0';
ClkMSB <= '0';
Wait for PERIOD * 8;

M1 <= '0';
M2 <= '1';
LR1245 <= '1';
Wait for PERIOD * 6;

Strmout <= '1';
Wait for PERIOD * 6;

VEnable <= '1'; HEnable <= '1';
LR1245 <= '0';
Strmout <= '0';
FZT <= '0';
S <= '0';
EnR0 <= '0';
ClkMSB <= '0';
Rev <= '0';
LR <= '0'; RC <= '1';
M1 <= '1'; M2 <= '1';
R0Ct1 <= '0'; R0Ct2 <= '1';
Pin <= '1'; Cin <= '0'; Sbin <= '0';
Lin <= '0'; Rin <= '0'; Uin <= '0'; Bin <= '0';
Abovei <= '0'; Belowi <= '0';
Wait for PERIOD;

VEnable <= '1'; HEnable <= '1';
LR1245 <= '0';
Strmout <= '0';
FZT <= '1';
S <= '0';
EnR0 <= '0';
ClkMSB <= '0';
Rev <= '0';
```

```
LR <= '0'; RC <= '1';
M1 <= '1'; M2 <= '1';
ROCtl <= '0'; ROCt2 <= '1';
Pin <= '1'; Cin <= '0'; Sbin <= '0';
Lin <= '0'; Rin <= '0'; Uin <= '0'; Bin <= '0';
Abovei <= '0'; Belowi <= '0';
Wait for PERIOD;

VEnable <= '1'; HEnable <= '1';
LR1245 <= '1';
Strmout <= '0';
FZT <= '0';
S <= '0';
EnRO <= '0';
ClkMSB <= '0';
Rev <= '0';
LR <= '0'; RC <= '1';
M1 <= '1'; M2 <= '1';
ROCtl <= '0'; ROCt2 <= '1';
Pin <= '1'; Cin <= '0'; Sbin <= '0';
Lin <= '0'; Rin <= '0'; Uin <= '0'; Bin <= '0';
Abovei <= '0'; Belowi <= '0';
Wait for PERIOD;

VEnable <= '1'; HEnable <= '1';
LR1245 <= '1';
Strmout <= '0';
FZT <= '1';
S <= '0';
EnRO <= '0';
ClkMSB <= '0';
Rev <= '0';
LR <= '0'; RC <= '1';
M1 <= '1'; M2 <= '1';
ROCtl <= '0'; ROCt2 <= '1';
Pin <= '1'; Cin <= '0'; Sbin <= '0';
Lin <= '0'; Rin <= '0'; Uin <= '1'; Bin <= '0';
Abovei <= '0'; Belowi <= '0';
Wait for PERIOD;


-- Test Process Ends Here

   Pin <= '1';
   wait;
end process T1;

END STIM;
```

Appendix B: ZTE VHDL Listings

```
-------------------------------------------------------------------------
-- IP ZTE STRUCTURAL DEFINITION                                        --
-- Authors G.Alagoda                                                   --
-- Version 003a (001 Was Crap)                                         --
-- For Single Pixel Test Only                                          --
-------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity PIXEL is
    port (Clk, Rst            : in  std_logic;
          RC                        : in  std_logic;
          LR                        : in  std_logic;
          LoadR0                    : in  std_logic;
          EnR0                      : in  std_logic;
          EnR1R2                    : in  std_logic;
          LoadR1R2_RZSym        : in  std_logic;
          Sub                       : in  std_logic;
          Hld_msb                   : in  std_logic;
          CycleR0                   : in  std_logic;
          AddC                      : in  std_logic;
          Streamout                 : in  std_logic;
          HP_Col_in                 : in  std_logic;
          HP_Row_in                 : in  std_logic;
          HP_Col_out                : out std_logic;
          HP_Row_out                : out std_logic;
          VEnable                   : in  std_logic;
          HEnable                   : in  std_logic;
          SymIO                     : in  std_logic;
          FZTR                      : in  std_logic;

          left_in                   : in  std_logic;
          right_in                  : in  std_logic;
          top_in                    : in  std_logic;
          bottom_in                 : in  std_logic;
          data_out                  : out std_logic;

          Sib_in  : in  std_logic;
          Cld_in  : in  std_logic;
          Par_in  : in  std_logic;
          Sib_out : out std_logic;
          Par_out : out std_logic
          );

end PIXEL;

architecture STRUCTURAL of PIXEL is

  -- Register set
       signal R0      : std_logic_vector(0 to 7);  -- 8-bit Register0
       signal R0_1    : std_logic;                 -- 1-bit Register0 appendage
       signal R1      : std_logic_vector(0 to 5);  -- 6-bit Register1
       signal R2      : std_logic_vector(0 to 5);  -- 6-bit Register2
       signal RZSym   : std_logic_vector (0 to 1); -- 1-bit Register4 For R0 & R1
       signal RZSig   : std_logic;
       signal R3      : std_logic;                 -- 1-bit Register3 Carry Store
       signal R6      : std_logic;                 -- 1-bit Register6 for FZtr

  -- muxes
       signal M1,M2,M3,M4,M5,M6,M7,M8,M9A,M9B,M9C,M10,M11,M12,M13,M14 : std_logic;

  -- De Muxes
       signal DM1_0, DM1_1 : std_logic;

  -- Adder
       signal S_Out, C_Out, C_In, T_Add : std_logic;

  -- Global Enable
       signal GEnable : std_logic;

  -- Zerotree Identification
```

```
        signal ZId : std_logic;

begin
        REG0: process (Rst, Clk)
        begin
                if Rst = '1' then
                        R0              <= "00000000";
                        R0_1    <= '0';
                elsif (Clk = '1' and clk'event) then
                        if ((not EnR0) and (M8 xor LR) ) = '1' then
                                R0_1 <= '0';
                        end if;
                        if (((M8 xor LR) nand (not EnR0)) and GEnable and (not SymIO)) =
'1' then
                                R0_1 <= R0(7);
                                if Hld_msb = '1' then
                                        R0(1 to 7) <= R0(0 to 6);
                                else
                                        R0(0 to 7) <= M6 & R0(0 to 6);
                                end if;
                        end if;
                end if;
        end process REG0;

        REG1: process (Rst, Clk)
        begin
                if Rst = '1' then
                        R1 <= "000000";
                elsif (Clk = '1' and Clk'event) then
                        if DM1_1 = '1' then
                                R1 <= M2 & R1(0 to 4);
                        end if;
                end if;
        end process REG1;

        REG2: process (Rst, Clk)
        begin
                if Rst = '1' then
                        R2 <= "000000";
                elsif (Clk = '1' and Clk'event) then
                        if DM1_0 = '1' then
                                R2 <= M1 & R2(0 to 4);
                        end if;
                end if;
        end process REG2;

        REG3: process (Rst, Clk, GEnable)
        begin
                if Rst = '1' then
                        R3 <= '0';
                elsif (Clk = '1' and Clk'event) then
                        if (GEnable) = '0' then
                                R3 <= '0';
                        else
                                R3 <= C_Out;
                        end if;
                end if;
        end process REG3;

        REG_SYM: process (Rst, Clk, GEnable, HEnable, VEnable)
        begin
                if Rst = '1' then
                        RZSym (0) <= '0';
                        RZSym (1) <= '0';
                else
                        if (Clk = '1' and Clk'event) then
                                if (GEnable and SymIO) = '1' then
                                        RZSym (0) <= M10;
                                        RZSym (1) <= M11;
                                end if;
                        end if;
                        if (VEnable nor HEnable) = '1' then
                                RZSym (0) <= '1';
                                RZSym (1) <= '1';
                        end if;
                end if;
        end process REG_SYM;
```

```
        REG6: process (Clk, GEnable)
        begin
                if (VEnable nor HEnable) = '1' then
                        R6 <= '1';
                elsif (Clk = '1' and Clk'event) then
                        if FZtr = '1' then
                                R6 <= (FZtr nand (Par_In or (RZSym (0) nor RZSym (1)) or
((not RZSym (0)) and RZSym (0) and (not SymIO))));
                        end if;
                end if;
        end process REG6;

-- Concurrent stuff

-- Muxes
        M1 <= S_Out when LoadR1R2_RZSym = '1' else R2(5);
        M2 <= S_Out when LoadR1R2_RZSym = '1' else R1(5);
        M3 <= R1(5) when Streamout  = '1' else R2(5);
        M4 <= S_Out when ((M8 nor EnR1R2) nor CycleR0) = '1' else M5;
        M5 <= R0(7) when LoadR0 = '1' else R0_1;
        M6 <= M9C when LoadR0 = '1' else M4;
        M7 <= (M3 and LR) when EnR1R2 = '1' else M9C;
        M8 <= HP_Col_In when RC = '1' else HP_Row_In;
        M9A <= right_in when LR = '1' else left_in;
        M9B <= bottom_in when LR = '1' else top_in;
        M9C <= M9B when RC = '1' else M9A;
        M10 <= (ZID or Cld_in) when LoadR1R2_RZSym = '1' else M9C;  -- Ztrea Id  '0's
because Al Screwed up diagram
        M11 <= (Cld_in or (Par_in and (not ZID))) when  LoadR1R2_RZSym = '1' else RZSym
(0);
        M12 <= M5 when SymIO = '0' else RZSym (0);
        M13 <= M12 when GEnable = '1' else M9C;
        M14 <= (Cld_In or ZID) when FZtr = '0' else (RZSym (0) nand RZSym (0));

-- De Muxes
        DM1_0 <= (EnR1R2 and LR) when Streamout = '0' else '0';
        DM1_1 <= (EnR1R2 and LR) when Streamout = '1' else '0';

-- Output Signal Definition
        Sib_Out <= (Sib_In or ZID or Cld_In);
        Par_Out <= M14;
        Data_Out <= M13;
        HP_Col_Out <= (HP_Col_In xor GEnable);
        HP_Row_Out <= (HP_Row_In xor GEnable);

-- Latches & Internal Signals
        GEnable <= (R6 and VEnable and HEnable and not(FZtr));
        RZSig <= (RZSig or (R0(6) xor R0(7))) and (not (not GEnable));
        ZID <= RZSig;

-- Serial Adder
        C_In <= R3 or AddC;
        T_Add <= (M7 xor Sub);
        S_Out <= (M5 xor T_Add) xor C_In;
        C_Out <= (T_Add and C_In) or (T_Add and M5) or (M5 and C_In);

end STRUCTURAL;
configuration CFG_PIXEL_STRUCTURAL of PIXEL is

  for STRUCTURAL

  end for;

end CFG_PIXEL_STRUCTURAL;

-------------------------------------------------------------
-- Test The Entire ZTE Pixel In One Shot Ver 001  --
-- Authors : A.Hassan, G.Alagoda                  --
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity IPTest is
end IPTest;

architecture STIMULUS of IPTest is
```

```
-- Time Setting
  constant PERIOD : time := 20 ns;

-- Component Defn
  component PIXEL
    port (
      Clk, Rst        : in  std_logic;
      RC              : in  std_logic;
      LR              : in  std_logic;
      LoadR0          : in  std_logic;
      EnR0            : in  std_logic;
      EnR1R2          : in  std_logic;
      LoadR1R2_RZSym  : in  std_logic;
      Sub             : in  std_logic;
      Hld_msb         : in  std_logic;
      CycleR0         : in  std_logic;
      AddC            : in  std_logic;
      Streamout       : in  std_logic;
      HP_Col_in       : in  std_logic;
      HP_Row_in       : in  std_logic;
      HP_Col_out      : out std_logic;
      HP_Row_out      : out std_logic;
      VEnable         : in  std_logic;
      HEnable         : in  std_logic;
      SymIO           : in  std_logic;
      FZTR            : in  std_logic;

      left_in   : in  std_logic;
      right_in  : in  std_logic;
      top_in    : in  std_logic;
      bottom_in : in  std_logic;
      data_out  : out std_logic;

      Sib_in  : in  std_logic;
      Cld_in  : in  std_logic;
      Par_in  : in  std_logic;
      Sib_out : out std_logic;
      Par_out : out std_logic;

      R0    : inout std_logic_vector(0 to 7);   -- 8-bit Register0
      R0_1  : inout std_logic;                  -- 1-bit Register0 Appendage
      R1    : inout std_logic_vector(0 to 5);   -- 6-bit Register1
      R2    : inout std_logic_vector(0 to 5);   -- 6-bit Register2
      RZSym : inout std_logic_vector (0 to 1);  -- 1-bit Register4 For B0 & B1
      RZSig : inout std_logic

      );
  end component;

-- Testbench Signal Definition
  signal Clk            : std_logic := '0';
  signal Rst            : std_logic;
  signal RC             : std_logic;
  signal LR             : std_logic;
  signal LoadR0         : std_logic;
  signal EnR0           : std_logic;
  signal EnR1R2         : std_logic;
  signal LoadR1R2_RZSym : std_logic;
  signal Sub            : std_logic;
  signal Hld_msb        : std_logic;
  signal CycleR0        : std_logic;
  signal AddC           : std_logic;
  signal Streamout      : std_logic;
  signal HP_Col_in      : std_logic;
  signal HP_Row_in      : std_logic;
  signal HP_Col_out     : std_logic;
  signal HP_Row_out     : std_logic;
  signal VEnable        : std_logic;
  signal HEnable        : std_logic;
  signal SymIO          : std_logic;
  signal FZTR           : std_logic;

  signal left_in   : std_logic;
  signal right_in  : std_logic;
  signal top_in    : std_logic;
  signal bottom_in : std_logic;
```

```
  signal data_out  : std_logic;

  signal Sib_in  : std_logic;
  signal Cid_in  : std_logic;
  signal Par_in  : std_logic;
  signal Sib_out : std_logic;
  signal Par_out : std_logic;

  signal R0    : std_logic_vector(0 to 7);
  signal R0_1  : std_logic;
  signal R1    : std_logic_vector(0 to 5);
  signal R2    : std_logic_vector(0 to 5);
  signal RZSym : std_logic_vector(0 to 1);
  signal RZSig : std_logic;

  signal Cnt : integer := 0;

begin
  IP : Pixel port map (Clk             => Clk, Rst => Rst, RC => RC, LR => LR, LoadR0 =>
LoadR0, EnR0 => EnR0, EnR1R2 => EnR1R2,
                       LoadR1R2_RZSym => LoadR1R2_RZSym, Sub => Sub, Hld_msb => Hld_msb,
CycleR0 => CycleR0, AddC => AddC,
                       Streamout      => Streamout,
                       HP_Col_in       => HP_Col_in, HP_Row_in => HP_Row_in, HP_Col_out
=> HP_Col_out, HP_Row_out => HP_Row_out,
                       VEnable         => VEnable, HEnable => HEnable, SymIO => SymIO,
FZTR => FZTR,
                       left_in            => left_in, right_in => right_in, top_in =>
top_in, bottom_in => bottom_in,
                       data_out       => data_out,
                       Sib_in         => Sib_in, Cid_in => Cid_in, Par_in => Par_in,
                       Sib_out        => Sib_out, Par_out => Par_out,
                       R0              => R0, R0_1 => R0_1, R1 => R1, R2 => R2, RZSym =>
RZSym, RZSig => RZSig
                       );

-- Concurrent setting for the clock
  Clk <= (not Clk) after (PERIOD/2);
  Cnt <= Cnt + 1   after PERIOD;

  STIM : process
  begin
    Rst     <= '1';
    VEnable <= '0';
    HEnable <= '0';
    wait for (PERIOD);
    Rst     <= '0';
    VEnable <= '1';
    HEnable <= '1';

    RC             <= '0';
    LR             <= '0';
    LoadR0         <= '0';
    EnR0           <= '0';
    EnR1R2         <= '0';
    LoadR1R2_RZSym <= '0';
    Sub            <= '0';
    Hld_msb        <= '0';
    CycleR0        <= '0';
    AddC           <= '0';
    Streamout      <= '0';
    SymIO          <= '0';
    FZTR           <= '0';
    left_in        <= '0';
    right_in       <= '0';
    top_in         <= '0';
    bottom_in      <= '0';

    Sib_in <= '0';
    Cid_in <= '0';
    Par_in <= '0';

    HP_Col_in <= '1';
    HP_Row_in <= '1';
-------------------------------------------------
-- Phase 01 - Pixel Load Mode
-- Load R0 = '00001110'          (14 dec)
```

```
-------------------------------------------------
    EnR0      <= '1';
    LoadR0    <= '1';
    LR        <= '0';
    RC        <= '0';

-- Load Data for R1 into R0
    left_in <= '0'; wait for PERIOD;
    left_in <= '1'; wait for PERIOD;
    left_in <= '1'; wait for PERIOD;
    left_in <= '1'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;

    assert (R0 = "00001110")
      report "Load Failed"
      severity failure;
-------------------------------------------------
-- Phase 02 - Motion Difference Mode
-- R0 = '00001110'      (14 dec)
-- R1 = '000010'        (00 dec) for test later
-------------------------------------------------
    EnR0    <= '1';
    CycleR0 <= '1';
    LoadR0  <= '0';
    Hld_msb <= '0';
    wait for PERIOD;

    CycleR0        <= '0';
    EnR1R2         <= '1';
    LoadR1R2_RZSym <= '0';
    Streamout      <= '1';
    Sub            <= '1';
    LR             <= '1';
    AddC           <= '1';
    wait for PERIOD;
    AddC           <= '0';
    wait for PERIOD;
    wait for PERIOD;
    wait for PERIOD;
    wait for PERIOD;
    wait for PERIOD;
    LR             <= '0';
    wait for PERIOD;
    wait for PERIOD;

    assert (R0 = "00001110") report "Motion Difference Fecked" severity failure;

    assert R1 = "000000" report "R1 loaded with crap from Motion Diff" severity failure;
-------------------------------------------------
-- Phase 03 - Forward Wavelet Mode
-- Left          = '00001111'     (15 dec)
-- Right         = '00001101'     (13 dec)
-- Top           = '00001010'     (10 dec)
-- Bottom        = '00010000'     (16 dec)
-------------------------------------------------
    HP_Col_In <= '1';
    HP_Row_In <= '1';
    EnR1R2    <= '0';
    LoadR0    <= '0';
    Hld_msb   <= '0';

-- Shift Right LP Pixels
    EnR0    <= '0';
    LR      <= '0';
    RC      <= '0';
    CycleR0 <= '1';
    wait for PERIOD;

-- Subtract from Left
    EnR0    <= '1';
    CycleR0 <= '0';
    RC      <= '0';
    LR      <= '0';
    Sub     <= '1';
```

```
    AddC    <= '1';
    left_in <= '1'; wait for PERIOD;
    AddC    <= '0';
    left_in <= '1'; wait for PERIOD;
    left_in <= '1'; wait for PERIOD;
    left_in <= '1'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;

    assert (R0 = "00000110" and R0_1 = '1')
      report "Sub from Left Buggered"
      severity failure;

-- Reset Carry Bit
    HEnable <= '0';
    VEnable <= '0';
    wait for PERIOD;
    HEnable <= '1';
    VEnable <= '1';

-- Subtract from Right
    RC       <= '0';
    LR       <= '1';
    AddC     <= '1';
    right_in <= '1'; wait for PERIOD;
    AddC     <= '0';
    right_in <= '0'; wait for PERIOD;
    right_in <= '1'; wait for PERIOD;
    right_in <= '1'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;

    assert (R0 = "00000000" and R0_1 = '0')
      report "Sub from Right Buggered"
      severity failure;

-- Rotate 7 All
    CycleR0 <= '1';
    wait for PERIOD*7;

-- Rotate 1 HP
    EnR0 <= '0';
    LR   <= '1';
    wait for PERIOD;

-- Shift Right LP Pixels
    RC <= '1';
    LR <= '0';
    wait for PERIOD;

-- Subtract from Above
    EnR0    <= '1';
    CycleR0 <= '0';
    RC      <= '1';
    LR      <= '0';
    Sub     <= '1';
    AddC    <= '1';
    top_in  <= '0'; wait for PERIOD;
    AddC    <= '0';
    top_in  <= '1'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '1'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;

    assert (R0 = "11111011" and R0_1 = '0')
      report "Sub from Top Buggered"
      severity failure;
```

```
-- Reset Carry Bit
    HEnable <= '0';
    VEnable <= '0';
    wait for PERIOD;
    HEnable <= '1';
    VEnable <= '1';

-- Subtract from Below
    RC         <= '1';
    LR         <= '1';
    Sub        <= '1';
    AddC       <= '1';
    bottom_in <= '0'; wait for PERIOD;
    AddC       <= '0';
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '1'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;

    assert (R0 = "11110011" and R0_1 = '0')
      report "Sub from Bottom Buggered"
      severity failure;
-------------------------------------------------
-- Phase 04 - Quantisation & Inverse (by Div 4)
-------------------------------------------------
    EnR1R2          <= '0';
    LoadR1R2_RZSym <= '0';
    Streamout       <= '0';
    Sub             <= '0';

    EnR0    <= '1';
    CycleR0 <= '1';
    LoadR0  <= '0';
    Hld_msb <= '1';
    wait for PERIOD*3;
    Hld_msb <= '0';

    wait for PERIOD*(9 - 3);

    assert (R0 = "11110011" and R0_1 = '1')
      report "Quantisation Barked"
      severity error;
-------------------------------------------------
-- Phase 05 - Significance Bit Generation
-------------------------------------------------
    HEnable <= '0';
    VEnable <= '0';
    wait for PERIOD;
    HEnable <= '1';
    VEnable <= '1';

    CycleR0 <= '1';
    LoadR0  <= '0';
    EnR0    <= '1';

    wait for PERIOD*9;

    assert (RZSig = '1')
      report "Significance screwed"
      severity error;
-------------------------------------------------
-- Phase 06a - Symbol Generation
-------------------------------------------------
    Sib_In          <= '0';
    Cld_in          <= '0';
    Par_in          <= '1';
    SymIO           <= '1';
    LoadR1R2_RZSym <= '1';
    wait for PERIOD;

    assert (RZSym = "10")
      report "Symbol shafted"
```

```
      severity error;
-------------------------------------------------------
-- Phase 07a - Symbol Extraction
-------------------------------------------------------
    LoadR1R2_RZSym <= '0';
    -- Form Zerotrees
    FZtr           <= '1';
    wait for PERIOD;
    FZtr           <= '0';
    RC             <= '0';
    LR             <= '0';
    left_in        <= data_out;
    wait for PERIOD;
    left_in        <= data_out;
    wait for PERIOD;
    left_in        <= data_out;

    assert (RZSym = "10")
      report "Symbol Reload rogered"
      severity error;
-------------------------------------------------------
-- Phase 07b - Coefficient Extraction
-------------------------------------------------------
    -- Form Zerotrees
    Par_in  <= '0';
    FZtr    <= '1';
    wait for PERIOD;
    FZtr    <= '0';
    RC      <= '0';
    LR      <= '0';
    SymIO   <= '0';
    LoadR0  <= '1';
    CycleR0 <= '0';
    left_in <= '1'; wait for PERIOD;
    left_in <= '1'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;

    assert (R0 = "00000011")
      report "Coefficient Reload ruined"
      severity error;
-------------------------------------------------------
-- Phase 08 - Inverse Wavelet Transform
-------------------------------------------------------
    HP_Col_In <= '1';
    HP_Row_In <= '1';
    EnR1R2    <= '0';
    LoadR0    <= '0';
    Hld_msb   <= '0';

-- Shift Right LP Pixels
    EnR0    <= '0';
    LR      <= '0';
    RC      <= '0';
    CycleR0 <= '1';
    wait for PERIOD;

-- Add from Left
    EnR0    <= '1';
    CycleR0 <= '0';
    RC      <= '0';
    LR      <= '0';
    left_in <= '1'; wait for PERIOD;
    left_in <= '1'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;
    left_in <= '0'; wait for PERIOD;

    assert (R0 = "00000100" and R0_1 = '1')
```

```
      report "Sum from Left hollocksed"
      severity error;

-- Reset Carry Bit
    HEnable <= '0';
    HEnable <= '0';
    wait for PERIOD;
    HEnable <= '1';
    HEnable <= '1';

-- Add from Right
    RC       <= '0';
    LR       <= '1';
    right_in <= '1'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;
    right_in <= '0'; wait for PERIOD;

    assert (R0 = "00000101" and R0_1 = '0')
      report "Sum from Right hollocksed"
      severity error;

-- Rotate 7 All
    CycleR0 <= '1';
    wait for PERIOD*7;

-- Rotate 1 HP
    EnR0 <= '0';
    LR   <= '1';
    wait for PERIOD;

-- Shift Right LP Pixels
    RC <= '1';
    LR <= '0';
    wait for PERIOD;

-- Add from Above
    EnR0    <= '1';
    CycleR0 <= '0';
    RC      <= '1';
    LR      <= '0';
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '1'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;
    top_in  <= '0'; wait for PERIOD;

    assert (R0 = "00001100" and R0_1 = '0')
      report "Sum from Top hollocksed"
      severity error;

-- Reset Carry Bit
    HEnable <= '0';
    HEnable <= '0';
    wait for PERIOD;
    HEnable <= '1';
    HEnable <= '1';

-- Add from Below
    RC        <= '1';
    LR        <= '1';
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '1'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;
```

```
    bottom_in <= '0'; wait for PERIOD;
    bottom_in <= '0'; wait for PERIOD;

    assert (R0 = "00001101" and R0_1 = '0')
      report "Sum from Bottom bollocksed"
      severity error;
-------------------------------------------------------
-- Phase 09 - Motion Summation Mode
-------------------------------------------------------
--              EnR0 <= '1';
--              CycleR0 <= '1';
--              LoadR0 <= '0';
--              Hld_msb <= '0';
--              wait for PERIOD;

    CycleR0       <= '0';
    EnR1R2        <= '1';
    LoadR1R2_RZSym <= '1';
    Streamout     <= '1';
    LR            <= '1';
    wait for PERIOD*6;
    LR            <= '0';
    wait for PERIOD*2;

    assert (R1 = "011010")
      report "Motion Summation crapped"
      severity error;
-------------------------------------------------------
    assert false
      report "Completed Successfully"
      severity error;
    wait;
  end process STIM;
end STIMULUS;

configuration CFG_IPTEST_STIMULUS of IPTEST is

  for stimulus
    for ip : pixel
      use configuration WORK.CFG_PIXEL_STRUCTURAL;
    end for;
  end for;

end CFG_IPTEST_STIMULUS;
```

This page is intentionally left blank.